

Microsoft®

Enterprise Development Reference Implementation

Version 1.1



patterns & practices

Enterprise Development Reference Implementation

Version 1.1

patterns & practices

Jason Hogg, Microsoft Corporation
Naveen Yajaman, Microsoft Corporation
Jim Newkirk, Microsoft Corporation
Jonathan Wanagel, Microsoft Corporation
Wojtek Kozaczynski, Microsoft Corporation
Ron Jacobs, Microsoft Corporation
Edward Lafferty, Microsoft Corporation
RoAnn Corbisier, Microsoft Corporation
Sameer Tarey, Infosys Technologies Ltd
Alejandro Guillermo Jack, Southworks S.R.L.
Lonnie Wall, RDA Corporation
Andrew Lader, RDA Corporation
Nelly Delgado, Wadeware LLC

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2004 Microsoft Corporation. All rights reserved.

Microsoft, MS-DOS, Windows, Windows NT, Windows Server, BizTalk, Microsoft Press, Visual Basic, Visual C#, and Visual Studio are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Contents

Chapter 1

Overview and Use Cases	1
Introduction	1
Intended Audience	2
Prerequisites	2
Global Bank Scenario	3
Enterprise Development Reference Implementation Community	5
Use of Existing Patterns and Practices	6
EDRI Documentation	6
Use Cases	7
Login	8
Consolidated Account Statement	10
Transfer Funds	12
Bill Subscriptions	14
Bill Payment	15
Transaction Log Report	16
Non-Functional Application Considerations	17
Application Availability	17
Data Integrity	17
Performance	18
Security	18
Browser Compatibility	21
Use Case Realization	21
Login and Consolidated Account Statement Use Cases	23
Transfer Funds Use Case	26
Bill Subscriptions Use Case	29
Add Bill Subscription	29
Delete Bill Subscription	32
Bill Payment Use Case	35
Transaction Log Report Use Case	39

Chapter 2

Architecture	41
Design Objectives and Principles	41
Objectives	42
Enterprise Framework	42
Service Oriented Integration	42
Architectural Representation	43
Terminology and Key Concepts	44

Conceptual View	45
Dominant Patterns	45
Model-View-Controller Pattern	45
Page Controller Pattern	46
Logical View	46
Model-View-Controller Pattern	46
Page Controller Pattern	48
Service Invocation	52
Implementation View	54
WebApplicationUI	55
WebApplication.UIProcess	56
Configuration	56
Deployment View	57
Architecture Properties View	58
Security	58
Localization	59

Chapter 3

Services

60

Service Infrastructure	61
Request Message Validation	61
Custom LoggingHandler	62
Service Interface and Service Implementation Pipelines	63
Web Service Headers	64
Common Design Views	64
Deployment View	65
Policy View	66
Service Documentation	67
Messages	67
Conceptual View	69
AccountStatement Service	70
Logical View	70
Client Interface	72
Deployment View	79
Authentication Service	80
Logical View	80
Client Interface	81
Deployment View	85
BillPayment Service	86
Logical View	86
Client Interface	88
Deployment View	93
BillSubscription Service	94
Logical View	94
Client Interface	95
Deployment View	101

FundsTransfer Service	101
Logical View	101
Client Interface	103
Deployment View.	107
TransactionLog Service	108
Logical View	108
Client Interface	109
Deployment View.	114
Summary	115
More Information.	115

Chapter 4

Installation 117

Introduction	117
Platform Prerequisites	117
Before You Begin.	118
Installing the Enterprise Development Reference Implementation	120
Verifying the Installation.	120
Setting Up and Running the EDRI	121
Using the Various Dispatching Transports.	123
To Uninstall.	125
Troubleshooting.	127
Troubleshooting Web Service Projects	127
Troubleshooting Set Up	127
More Information.	129

Appendix A

Inside the Enterprise Development Application Framework 131

Introduction	131
Architectural Goals and Prerequisites	131
Goals	131
EDAF at a Glance	132
Dominant Patterns	136
Service Request Flow	139

Appendix B

Exploring the EDAF Using the Bill Payment Use Case 143

Contributors 150

Additional Resources 153

1

Overview and Use Cases

Introduction

Welcome to the Microsoft® Enterprise Development Reference Implementation (EDRI), version 1.1. This reference implementation is based on a fictitious banking business named Global Bank. The EDRI illustrates some of the challenges common to a wide variety of organizations, not just retail banks. This release of the EDRI is designed to serve three main purposes:

- Demonstrate the use of *patterns & practices* guidance within a reference implementation.
- Demonstrate the use of the Enterprise Development Application Framework (EDAF).
- Establish a community that influences the development of new business scenarios and the selection of appropriate technologies.

Note: The EDRI is not intended as a template for an online banking application, and does not solve all the challenges that would be faced in a real implementation.

This release of the EDRI also illustrates the implementation of a loosely coupled application. It demonstrates the use of design patterns, such as Page Controller and Model-View-Controller, within the Presentation tier, as well as service-oriented integration using ASP.NET Web services, and the recently released EDAF within the Business Services tier.

Version 1.1 of the EDRI is not intended to solve all problems relating to the development of a loosely coupled application. For example, this release does not cover securing Web services. However, Microsoft intends to provide regular updates to this scenario that address this and other challenges. Future releases of the EDRI may address other challenges including:

- Guidance for using WS-Security and Web Services Enhancements (WSE 2.0) to secure the application.
- Demonstration of the use of the [User Interface Process \(UIP\) Application Block – version 2.0](#).
- Implementation of a [Windows Forms](#) client with offline capabilities.
- Localization support.
- Implementation of key integration patterns demonstrating business process orchestration using BizTalk[®] Server 2004.
- Development guidance to illustrate the tenets of service orientation.
- Deployment scenarios using Virtual PC 2004 and Virtual Server 2005 in a Microsoft Solution Architecture environment.

Your feedback will guide the evolution of this program. For more information about how you can contribute, see the “Enterprise Development Reference Implementation Community” section.

Intended Audience

The Enterprise Development Reference Implementation demonstrates development of a loosely coupled Internet banking application for architects and lead developers who are interested in the following:

- Seeing *patterns & practices* guidance applied.
- Learning more about how to apply the EDAF.

Prerequisites

Because the reference implementation is built using the EDAF, you must install the framework. See Chapter 7, “Installation and QuickStarts” in the Enterprise Development Reference Architecture (EDRA) documentation for details. You can download the **Enterprise Development Reference Architecture.msi**. file from the [community workspace](#).

Additionally, the EDRI and EDAF use Microsoft .NET technologies, so it is helpful to be familiar with the Microsoft .NET Framework, Microsoft Visual Studio[®] .NET 2003 development system, and either the Microsoft Visual C#[®] or Microsoft Visual Basic[®] development systems.

Note: The EDRI and EDAF require either the Enterprise Architect edition or Enterprise Developer edition of Microsoft Visual Studio .NET 2003 because these editions have enterprise template support. See Chapter 4, “Installation” for additional platform prerequisites.

Some familiarity with the issues and challenges of developing distributed enterprise solutions is required. For more information about designing distributed applications, see [Application Architecture for .NET: Designing Applications and Services](#).

Finally, many principles described in *patterns & practices* guidance have been applied during the development of the Enterprise Development Reference Implementation. A complete list of these topics is available in “Use of Existing Patterns and Practices.”

Global Bank Scenario

Global Bank is a midsize, traditional bank that acquired a complete range of financial services capabilities through a series of acquisitions. Global Bank’s systems and supporting technologies evolved over time at different rates. Due to these acquisitions, the fragmentation of technology within its divisions, and the natural evolution of technology, Global Bank struggled to establish an online presence.

Global Bank’s executive management has decided to expand its online capabilities by offering customers access to a full range of services through an Internet banking application. In addition to visiting Global Bank’s brick and mortar locations, customers will be able to visit the Internet banking application to transfer funds, view account information, request reports, obtain financial advice, create financial plans, or access other services that Global Bank offers. Not all of these features are available in this release of the EDRI.

Like many large organizations, Global Bank has a diverse range of back-end technologies that support the day-to-day operations of the bank. Many of these technologies run on proprietary software and hardware with limited capability for reuse. As a move toward broader adoption of a service oriented architecture, Global Bank is investigating extending its functional integration strategy to incorporate standards-based Web services.

Figure 1 illustrates key systems that the Global Bank Internet banking application will interact with. Future updates to the Voice Response System and the Smart Client Teller Applications plan to use services developed in this release to optimize functional integration.

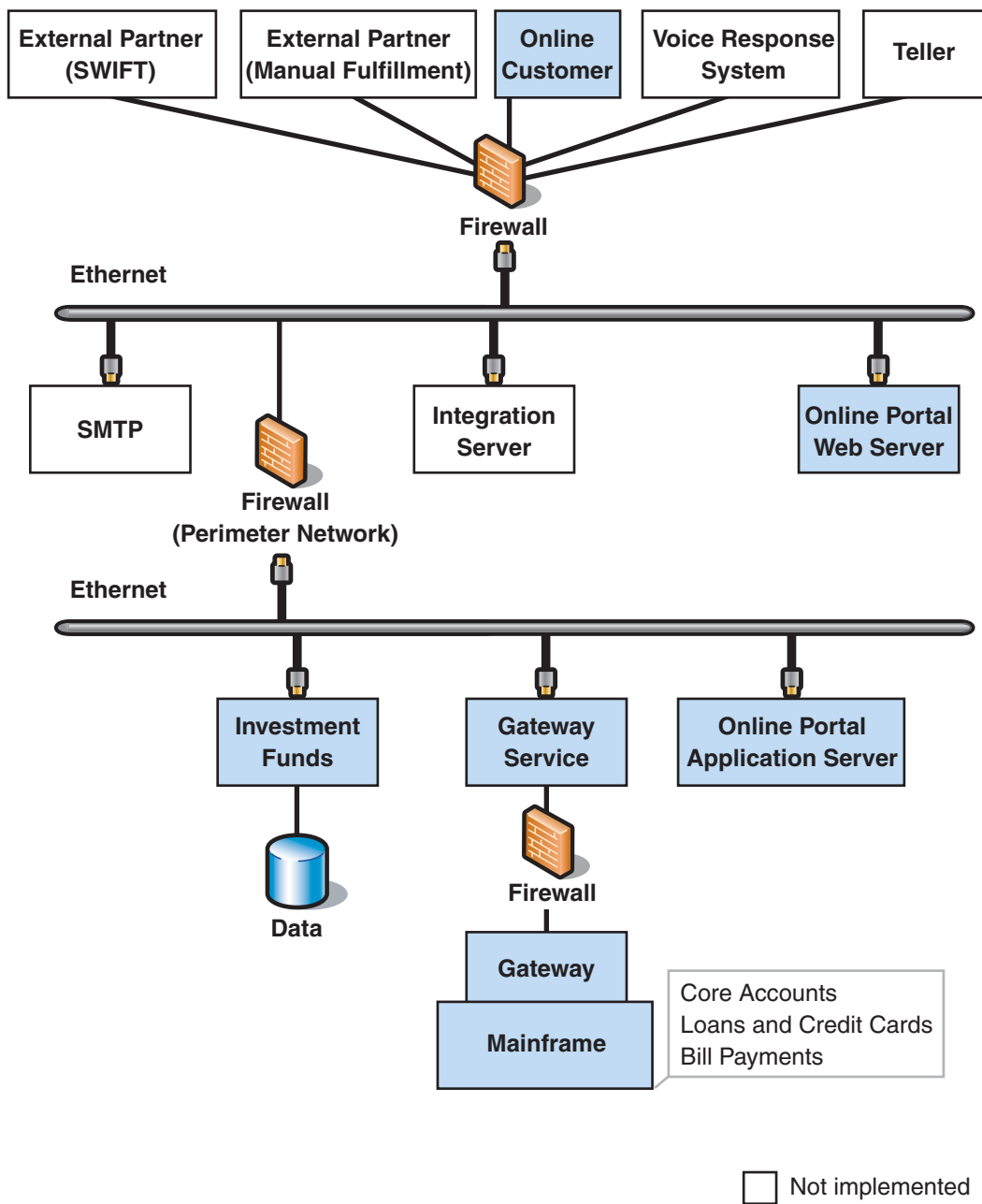


Figure 1
Enterprise network diagram

The EDRI will allow customers to conduct one stop banking across products such as the following:

- **Core accounts.** This includes savings and checking accounts.
- **Lending.** This includes credit cards, consumer loans (such as, auto and line of credit), mortgage, and home equity.
- **Investing.** This includes certificates of deposit, trust services, brokerage services (including securities), annuities, individual retirement accounts, and mutual funds.
- **Financial planning.** This includes comprehensive financial planning services, including retirement, education, tax, and estate planning, including both future planning and plan execution services.

Note: Some features in the preceding list are available in EDRI version 1.1. Other features are out of scope but could be addressed in another implementation of the Global Bank scenario.

Enterprise Development Reference Implementation Community

The EDRI was designed with the Microsoft developer community's participation in mind. The *pattern & practices* team has been using a community workspace as a way for the community members to review the EDRI and to provide feedback.

We encourage you to join the community at our workspace, where you can ask questions, get answers and share your ideas.

To get involved,

- Visit the [EDRA community workspace](#).
- Visit the [EDRA Wiki](#).
- Visit the [EDRI Wiki](#).

Use of Existing Patterns and Practices

An important goal in developing the GBRI was to demonstrate usage of *patterns & practices* guidance, including the following:

- *patterns & practices* and industry standard design patterns, including the following:
 - [Model-View-Controller](#)
 - [Page Controller](#)
- *patterns & practices* [Enterprise Development Application Framework](#), which in turn uses *patterns & practices* application blocks, including the following:
 - [Authorization and Profile Application Block](#)
 - [Configuration Management Application Block](#)
 - [Data Access Application Block](#)
 - [Logging Application Block](#)
- Guidance from *patterns & practices* books on designing secure, scalable, and performant applications, including the following:
 - [Improving Web Application Security: Threats and Countermeasures](#), Redmond: Microsoft Press, 2003, ISBN: 0735618429
 - [Building Secure Microsoft ASP.Net Applications: Authentication, Authorization and Secure Communication](#), Redmond: Microsoft Press, 2003, ISBN: 0735618909
 - [Improving .NET Application Performance and Scalability](#), Redmond: Microsoft Press, 2004, ISBN: 0735618518

EDRI Documentation

The remainder of this chapter covers the following topics:

- **Use Cases.** This section presents use cases that describe the features and functional requirements that are implemented in the EDRI.
- **Non-Functional Application Considerations.** This section describes the non-functional requirements that were considered in the development of the EDRI.
- **Use Case Realization.** This section provides a story board description (through screen shots and sequence diagrams) of each use case.

The remaining chapters are as follows:

- **Chapter 2, Architecture.** This chapter discusses the design objectives and principles as well as the conceptual, logical, implementation, and deployment view for the Enterprise Development Reference Implementation.
- **Chapter 3, Services.** This chapter presents a conceptual, logical, and deployment view to illustrate each of the EDRI services.
- **Chapter 4, Installation.** This chapter describes the required platform prerequisites and the steps you should follow to install the EDRI.
- **Appendix A — Enterprise Development Application Framework Overview.** This section gives a brief introduction to the EDAF.
- **Appendix B — Exploring the EDAF Using the Bill Payment Use Case.** This section gives an example of how the EDRI uses the EDAF.

Use Cases

The Enterprise Development Reference Implementation is a partial realization of a business scenario that incorporates challenges common to a wide variety of organizations — it does not apply only to banks. The EDRI functional requirements were determined through the process of analyzing the problem statement and forming use cases. A use case is a description of a system behavior observable by a user. This release concentrates on the following six use cases:

- Login
- Consolidated Account Statement
- Transfer Funds
- Bill Subscriptions
- Bill Payment
- Transaction Log Report

In many cases, only parts of the use case are implemented. This allows us to focus on challenging scenarios instead of the more routine aspects of the system that would normally be developed in such an application. Additional use cases may be implemented in future implementations of the Global Bank scenario. The parts of each use case that are not implemented are noted as “out of scope” within the discussion of each use case. The following sections describe in more detail the use cases the EDRI implements.

Login

The Login use case allows the Global Bank's Internet Banking Application to identify customers. The system prompts the user to enter his or her account number and password. Each login attempt is logged for auditing purposes. The user is authenticated after the system successfully identifies the user. The system's transactions and customization options are available only to authenticated users. Table 1 provides details of the Login use case.

Table 1: Login Use Case

Actors	Customer, System
Pre-Conditions	None
Actions	<ol style="list-style-type: none">1. The customer enters his or her account number and password on the home page and clicks the Go button.2. If the supplied credentials match those stored in the Customer Information database, the system presents the Consolidated Account Summary report. For more information, see the Consolidated Account Summary report use case.3. The system shows a personalized welcome message that includes the customer's name and last login date.4. The auditing system logs each login attempt including the following information: customer name, channel, last login date, and context information indicating whether the login was successful.
Alternative Flows	<p>Login on home page fails If the supplied credentials on the home page are incorrect, the system redisplay the login screen. The customer may then reenter his or her credentials.</p> <p>Login attempts and expiration If the customer unsuccessfully attempts to login more than five times, the customer's account is disabled. The customer will need to speak with a bank representative to reactivate the account.</p> <p>Login sessions are valid for 20 minutes. If the customer's session is inactive for more than 20 minutes, any attempt to access restricted information results in a redirection to the Login screen. The customer should open a new browser session.</p>

(continued)

Table 1: Login Use Case (continued)

Actors	Customer, System
Alternative Flows (out of scope)	<p>First time logins</p> <p>When a customer logs in to the system for the first time, the system prompts the customer to change his or her password before proceeding to the next page. This occurs when he or she first subscribes to the system or forgets the password and requests a new one.</p> <p>Change password request</p> <p>Customers may change their password. Passwords must comply with the system password policy. All attempts to change the password are logged in the Auditing and Security System.</p> <p>Customer forgets the account number or password</p> <p>If the customer forgets the account number or password, he or she needs to contact the Help Desk for assistance. In the case of the password, the Help Desk will assign a new password to the customer. After logging in with the newly assigned password, the system prompts the customer to change his or her password.</p>
Post-Conditions	None
Business Rules (out of scope)	Passwords must have a minimum length of four digits and a maximum length of eight digits. No more than two digits may appear consecutively. No more than two consecutive digits should appear in the sequence.
Comments	The 8-digit account number would likely be insufficient for a real bank over a large number of years.

Consolidated Account Statement

The Consolidated Account Statement use case describes the process of generating an on-demand, consolidated report summarizing the customer's contracts with the bank. It includes checking and savings accounts, credit cards, term deposits, investment funds, and pending bills. Table 2 provides details of the Consolidated Account Statement use case.

Table 2: Consolidated Account Statement Use Case

Actors	Customer, System
Pre-Conditions	The customer must be authenticated. Anonymous requests are forbidden.
Actions	<ol style="list-style-type: none"> 1. The customer selects the Account Summary option in the Global Bank online banking portal. 2. The system presents a summarized report of all contracts between the customer and the bank. Customers may expand each product to obtain more information (out of scope). See Table 3 for details. 3. A report displays the information grouped by the following sections: cash accounts, investment funds, cumulative deposits, credit cards, and pending bills. 4. The system displays the groups in the order specified in step 3, except when the customer's personalization options indicate otherwise (out of scope). 5. The system displays the information within each group. For details, see Table 3.
Alternative Flows	<p>Customer does not have products in a particular section</p> <p>If the customer does not own any products in a particular section, the system does not display the whole section. For example, if he or she does not have term deposits, the Investments section is not presented. The only exception is pending bills. If the customer subscribes to the Bill Payment system and does not have any pending bills, the system displays "No pending bills to display." If the customer does not subscribe to the Bill Payment system, the entire section does not appear.</p>
Post-Conditions	None
Business Rules	<p>The account summary option is available only to authenticated users.</p> <p>A customer may request a report only on his or her products.</p>

Table 3: Account Product Detail Information

Product	Fields	Description
Checking and Savings Account	Account Type	Account type
	Account Number	Account number
	Balance (USD)	USD available balance
Credit Cards	Card Number	Card number
	Card Type	Master Card or Visa
	Balance (USD)	Current balance
	Credit Limit (USD)	Credit limit on the card
	Expiration Date	Date card expires
	Payment Due Date	Date current payment is due
	Name on card	Customer name as it appears on the card
Cumulative Deposits	Identifier	Identifier
	Principal	Principal amount
	Interest	Interest accrued at the end of the term
	Term	Number of days
	Action on due date	Action may be: automatic renewal, deposit
	Due Date	Date the cumulative deposit is due
Investment Funds	Name	Name of the fund
	Shares	Number of shares (units) owned
	Quote	Quote for each share (unit)
Bills	Bill Payee	The name of the company
	Bill	Identifier for the bill
	Due Date	Date the bill is due
	Amount (USD)	The bill amount

Transfer Funds

The Transfer Funds use case allows customers to transfer money between accounts. Table 4 provides details of the Transfer Funds use case.

Table 4: Transfer Funds Use Case Details

Actors	Customer, System
Pre-Conditions	The customer must be authenticated. Anonymous requests are forbidden.
Actions	<ol style="list-style-type: none"> 1. The customer selects the Funds Transfer option in the Global Bank online banking portal. 2. The customer enters the source account from a list containing all of his or her savings and checking accounts, the destination account, the amount of money to transfer and a comment for future reference. The listing of multiple configured accounts and multiple currencies is out of scope. 3. The system prompts the customer for transaction confirmation. This step ensures that the destination account is eligible for this transaction (account is active and exists). 4. The system displays a unique identifier of the operation for the customer's future reference and the transfer is complete.
Alternative Flows	Transfer limits The amount of money that can be transferred in a single transaction is set across the system and can be configurable by Global Bank.
Alternative Flows (out of scope)	Transfer limits There will be a maximum amount of money that can be transferred in a month. The monthly limit applies to all transfers performed by the customer regardless of the account type. If the limit is exceeded, an error message is presented and the transfer is canceled. Transfers between accounts of different currencies Limits are expressed in the debiting account currency. If the debit and credit currencies differ, a foreign currency exchange transaction is triggered. The system displays a foreign exchange transaction confirmation for the customer's approval. Upon confirmation, the system displays the exchange rate used to perform the transaction. The system will log one entry for the exchange and another entry for the transfer to the Auditing and Security System. Third-party transfer limits If the destination account is not owned by the customer, a different system-wide limit applies to the maximum transfer amount. If the limit is exceeded, an error message is displayed and the transfer is canceled.

(continued)

Table 4: Transfer Funds Use Case Details (*continued*)

Actors	Customer, System
Alternative Flows (out of scope) (<i>continued</i>)	<p>Scheduled transfers</p> <p>By default, transfers take effect immediately after confirmation. The customer may defer the transfer by entering an effective date. The scheduled transaction is performed at the bank's opening time on the selected date. The system logs one entry when the customer requests the transfer and another entry when the transfer occurs to the Auditing and Security System.</p> <p>Transfers to any account can be scheduled. If the amount transferred exceeds the limit, an error message is displayed and the transfer is canceled.</p>
Post-Conditions	<p>Available balances</p> <p>Balances are updated after the operation is confirmed. If the customer requests a Consolidated Account Statement, the updated balances are displayed. The only exception is when the customer requests a scheduled transfer.</p>
Business Rules	<p>A customer can transfer funds from only his or her savings or checking accounts.</p> <p>The system will not allow duplicate transactions. If the user confirms an operation twice, the operation should not produce two transactions.</p> <p>Funds from the source account will be debited immediately after confirmation. If there are not enough funds to perform the transaction, the system will display an error message and return to the previous screen.</p> <p>The source and destination accounts for the funds transfer must be different.</p>
Business Rules (out of scope)	<p>The system will allow the maximum transfer amount to be set by the system administrator.</p> <p>If the currencies of a debit and credit amount during a funds transfer differ, the system will trigger a foreign currency exchange transaction.</p> <p>If the system triggers a foreign currency exchange during a transfer funds transaction, the system will display the exchange rate used to the customer during the confirmation process.</p> <p>Scheduled transfers will be automatically executed by the system at the bank's opening time on the scheduled date.</p>
Comments	Data entered on the page is validated both on the client and in the service.

Bill Subscriptions

The Bill Subscriptions use case allows customers to subscribe to bill payments. New bills coming from the subscribed external company are presented to the customer for payment. Table 5 provides details of the Bill Subscription use case.

Table 5: Bill Subscriptions Use Case

Actors	Customer, System
Pre-Conditions	The customer must be authenticated. Anonymous requests are forbidden.
Actions	<ol style="list-style-type: none"> 1. The customer selects the Bill Subscription option in the Global Bank online banking portal. 2. The system displays a list of the customer's subscribed bills. 3. The customer selects a bill payee from the list and the enters his or her account number for the selected payee. 4. The customer confirms the request and the new subscription is added to the list. 5. The system displays a unique identifier of the operation and the Bill Subscription — Receipt page.
Alternative Flows	<p>Delete subscription</p> <p>Each row on the subscribed bills section will have a delete button. If the customer clicks this button, the bill subscription is set to inactive. This will be allowed only if there are no pending bills to be paid.</p>
Alternative Flows (out of scope)	<p>Bill identifier validation</p> <p>The customer enters a Bill Identifier that identifies the relationship between him or her and the external company. If the external company provides this service to the bank, validations will occur.</p> <p>Billing Company Notification</p> <p>The billing company may be notified of the subscription. Notifications will happen asynchronously, meaning that they will not affect the subscription transaction. The EDRI does not show processing that would normally occur in an offline batch mode.</p>
Post-Conditions	<p>Cached Account Summary Information</p> <p>The cached Account Summary in the Web application is automatically updated. The Bill Subscription service returns the state of the accounts after subscribing the bills. This information is used by the Web application to update the cache.</p>

Bill Payment

The Bill Payment use case allows customers to pay bills. Table 6 provides details of the Bill Payment use case.

Table 6: Bill Payment Use Case

Actors	Customer, System, External Company, Bill System
Pre-Conditions	The customer must be authenticated. Anonymous requests are forbidden.
Actions	<ol style="list-style-type: none"> 1. The customer selects the Bill Payment option in the Global Bank online banking portal. 2. The customer selects an account from a list of his or her checking or savings accounts. 3. The customer selects one of the pending bills from a list of pending bills. 4. The system prompts the customer for transaction confirmation. 5. The system prompts the customer to confirm the request. 6. The system completes the transaction and displays the Bill Payment page.
Alternative Flows	Billing Company Notification The payment process implies a funds transfer operation between the customer account and a collection account belonging to the external company. The company may request to be notified and validate the transaction as part of the payment. This notification will be managed through a business event queue. If the publish fails, the implicit funds transfer operation must be reversed. Each of these events is logged.
Alternative Flows (out of scope)	Bill Company Notification, continued Subscription to the business event queue and notification to the bill payee is out of scope.
Post-Conditions	Cached Account Summary Information The service that performs the bill payment returns the state of the accounts after paying the bills. The Web application uses this information to update the cached information.
Business Rules	<p>After a bill is paid, it will not be shown in the account summary screen.</p> <p>The system will perform a check against a system-wide limit for each bill. Bills exceeding the limit will not be eligible and will be rejected.</p> <p>Limits on fund transfers will not affect and are not affected by bill payments.</p> <p>Funds from the source account will be debited immediately upon confirmation. If there are not enough funds to perform the transaction, the system will display an error message and return to the bill payment initial screen.</p> <p>The customer will not have the ability to select bills with amounts exceeding the account balance. This will be updated if the customer modifies the payment account.</p>

Transaction Log Report

The Transaction Log Report use case allows customers to view a log of selected transactions using a search criteria based on date ranges. Table 7 provides details of the Transaction Log Report use case.

Table 7: Transaction Log Report Use Case

Actors	Customer, System
Pre-Conditions	The customer must be authenticated. Anonymous requests are forbidden.
Actions	<ol style="list-style-type: none"> 1. The customer selects the Transaction Log option in the Global Bank online banking portal. 2. The system will show all transactions done on the selected date, including funds transfers and bill payments. 3. The system displays a list of records. The records contain information regarding the original operation including bill payee, account information, amount, and date.
Alternative Flows	Custom Report The customer selects a range of dates for a report using the Transaction Log Report. Only records for the previous three months are presented.
Alternative Flows (out of scope)	Logout Report When the customer logs out of the Global Bank online banking portal, the system prompts the customer for an optional Transaction Log Report. This option is a customization option and the customer may activate or deactivate it.
Post-Conditions	None
Business Rules	If the customer selects the custom date range, the “To” date must be greater than the “From” date.
Comments	The business rule validation should be done both on the client and in the service.

Non-Functional Application Considerations

In addition to the functional requirements discussed within the use cases, Global Bank also considered non-functional requirements related to areas such as availability, data integrity, performance, and security. Many of these issues are out of scope for this version.

Application Availability

The CIO is willing to allocate infrastructure dollars to support four nines (99.99%) of planned availability. Meeting this goal will require careful design of both the network and application architecture. Single points of failure within the infrastructure should be reduced by deploying redundant network switches, firewalls, application servers, and database servers.

The Internet banking application and dependent services should be designed so that additional servers can be added based on increases in traffic. The Presentation tier should be designed to scale horizontally, meaning that additional Web servers can be added to the Web farm as traffic increases.

Session state within the Presentation tier should be stored on a central database server to ensure users' sessions are not lost if a Web server goes offline. Web services running on the application servers should not maintain session state. This simplifies horizontal scaling, and it ensures service and data availability if an application server goes offline. To help support autonomy of deployed services, all services should react appropriately when dependent resources are unavailable.

Database servers should be deployed within a cluster to ensure availability if the database server goes offline. Databases can be designed to scale either vertically or horizontally. Most services in this fictitious scenario interact with applications which would, in real life, need to be tested to determine the extent to which they could meet the CIO's availability objectives.

"Stand in balance" services are often used to provide offline information when any back-end system becomes unavailable. For instance, if the mainframe is running batch processes and a customer requests an account summary, a handler routes the request to a different business action that reads the data from a stand-in database. This is out of scope.

Data Integrity

Autonomous services should handle failure of the client or service infrastructure without notification. System integrity should be ensured using techniques such as transactions, durable queues, redundant deployment, and failover.

Each service should describe the techniques that are being used to ensure system integrity. Examples include atomic transactions, compensating transactions, and message queues. Data integrity considerations include the following:

- A user may view or perform transactions only for accounts that they own.
- The system should not allow duplicate transactions. For example, if the user confirms the same operation twice, only one transaction should be produced.
- The system should not be affected by threats such as URL changes, hidden field hacks, or spoofing.
- Each Web service should validate messages to ensure adherence to WSDL contract and XSD schema.
- SSL and IPsec can also be used to ensure messages are not modified in transit. For pointers to information about how to configure SSL and IPsec, see the “Enterprise Development Reference Implementation Installation Guide.”
- The system should log the last date and time when any data is added, modified, or deleted; and it should log the user name that added, modified, or deleted the data.
- Passwords and credit card numbers should be transmitted by way of SSL, and the credit card numbers should be protected in the database. This is not supported in version 1.1.

Performance

Global Bank has referenced online banking application response time statistics available from [Keynote](#) and [Gomez](#). Global Bank has set an average page response time goal of 10–15 seconds (depending on site traffic).

This release of Global Bank has undergone performance and scalability testing, within a laboratory environment, to ensure this goal can be met. However, a later release of this solution may (depending on customer feedback) include a complete performance model and a sample capacity plan based on prescribed guidance. For end-to-end guidance for managing performance and scalability throughout your application life style, see [Improving .NET Application Performance and Scalability](#).

Security

The application and Web service implementations described in this document would, in reality, rely on a combination of trusted subsystem security for authentication and transport-level security for encryption and data integrity. A future release of Global Bank may demonstrate message level security using Web Services Security (WS-Security) as implemented by Web Services Enhancements (WSE) for Microsoft .NET. For more information about the trusted subsystem model, see [Chapter 3: Authentication and Authorization](#) in *Building Secure Microsoft ASP.NET Applications: Authentication, Authorization, and Secure Communication*.

Note: EDRI version 1.1 is set up for a development environment; it is not set up for a production environment. It is up to the developer to implement security based on their organizational and application specific requirements. You may also want to review the “Security Considerations” topic in the Enterprise Development Reference Architecture documentation.

For complete guidance on security issues, see the following resources:

- *Building Secure Microsoft ASP.NET Applications: Authentication, Authorization, and Secure Communication*, Redmond: Microsoft Press, 2003
 - *Improving Web Application Security: Threats and Countermeasures*, Redmond: Microsoft Press, 2003
-

User Authentication

User identification and passwords will be obtained using ASP.NET forms authentication. SSL would, of course, be used to ensure confidentiality of user data during authentication — this is not configured, by default, when the application is installed.

User credentials will be validated, and a generic principal containing the end-user’s identity will be created and stored in the session state on the Web servers. This allows interactions with back-end services to validate data entitlement of the originating user.

Global Bank has two future requirements that will impact how end-user credentials are managed in the long term:

- **Support for additional channels.** A future update of the Voice Response System and proposed Smart Client applications that tellers use will consume the same Web services as those developed for the online banking application. At that point, the Web services will be exposed to more than one client, making sole reliance on a trusted subsystem model a little more complex.

To provide a greater degree of security when the services are exposed to more than one client, Global Bank is considering developing a service that would issue tokens when an end user is initially authenticated. This token would be stored in a transient token database that will be keyed by session ID. The table would also contain the user’s identity and other information related to token expiration. This is not implemented in this release.

- **Client authentication using X509 certificates.** Global Bank would like to investigate options for authenticating end users using solutions such as X509 certificates; however, this capability is not widely used in the United States where Global Bank is currently based.

Additional user identification and authentication considerations include the following:

- The system should uniquely identify a user with a user account and password combination before allowing him or her to access the system. The user account must be a minimum of eight characters.
- After users are logged in, they will automatically be logged out if the session is idle for 20 minutes. Subsequent requests to protected pages will force the user to reenter his or her credentials at the Login page.
- The system should allow the user to change his or her password at any time. This is not implemented in this release.

User Authorization

Global Bank should enforce the following functional entitlement requirements:

- The Login page can be accessed by any user.
- The Account Summary, Funds Transfer, Bill Subscription, Bill Payment, and Online Help pages can be accessed only by authenticated users.
- Data entitlement should also be implemented to ensure that a user can view his or her own data.

Service Authentication

In this release of Global Bank, the Presentation tier will be the only client using the Web services. Network level security, such as firewalls and private IP addresses, should be used to restrict access to these services.

Authentication between the Presentation tier and the application server will rely on a trusted subsystem model where the application server validates the calling server's credentials, not those of the end user initiating the request. See "The Trusted Subsystem Model" in [Chapter 3: Authentication and Authorization](#) in *Building Secure Microsoft ASP.NET Applications: Authentication, Authorization, and Secure Communication*.

Transport Layer Security

Interactions between client and Presentation tier are assumed to occur using SSL/TLS with server certificates. Security between the Presentation tier and the application servers hosting the business logic includes network level security using SSL to encrypt the transport. IPSec will also be used to secure the data transported between the Presentation tier and the Application tier. SSL client certificates are also being considered as an additional means of allowing the application server to validate the credentials of the Presentation tier.

Browser Compatibility

The system should take multiple browsers into consideration. For this release, the EDRI has been tested only against Microsoft Internet Explorer 6.0 or later.

Use Case Realization

The Global Bank Internet banking application presents the user with a Web interface to the Global Bank functionality. The number of pages is small in this case, because only a few functional areas are available:

- Login
- Consolidated Account Statement
- Transfer Funds
- Bill Subscriptions
- Bill Payment
- Transaction Log Report

The number of use cases being implemented is a subset intended to demonstrate complex problems. This section provides a story board description of each use case. The story board walks you through the user interface that implements the use case. Each use case also contains a sequence diagram that describes the interaction between users, systems, and services used to implement the use case. Implementation of the presentation tier and back-end Web services is described in detail in Chapter 2, “Architecture” and Chapter 3, “Services.”

Figure 2 presents the Global Bank Internet banking application's screens and navigation paths.

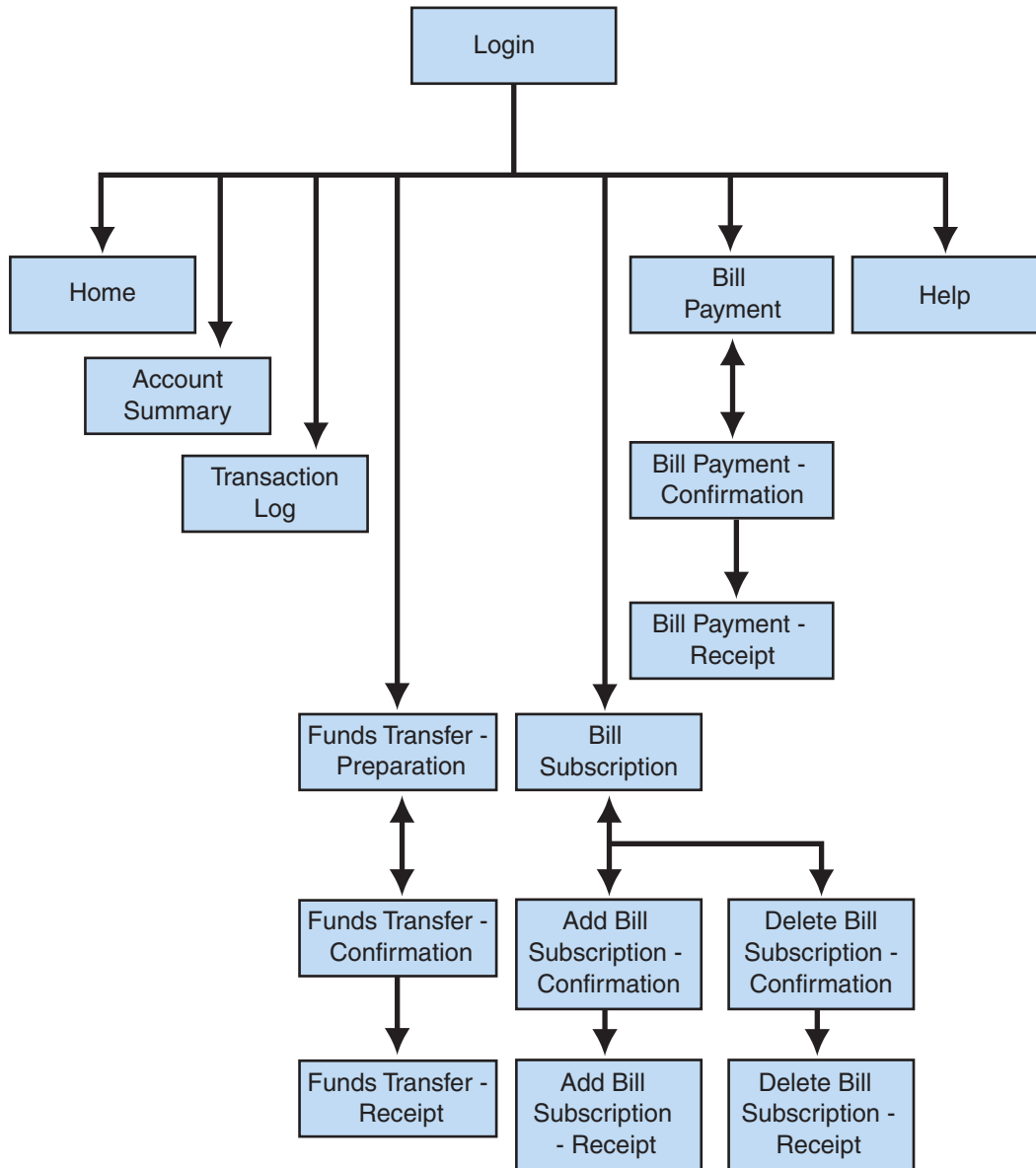


Figure 2

Global Bank navigation map

Login and Consolidated Account Statement Use Cases

The Login page is the first screen in the Global Bank Internet banking application. Each user will enter his or her user name and password, and then clicks the **Go** button. Figure 3 shows the Login page.

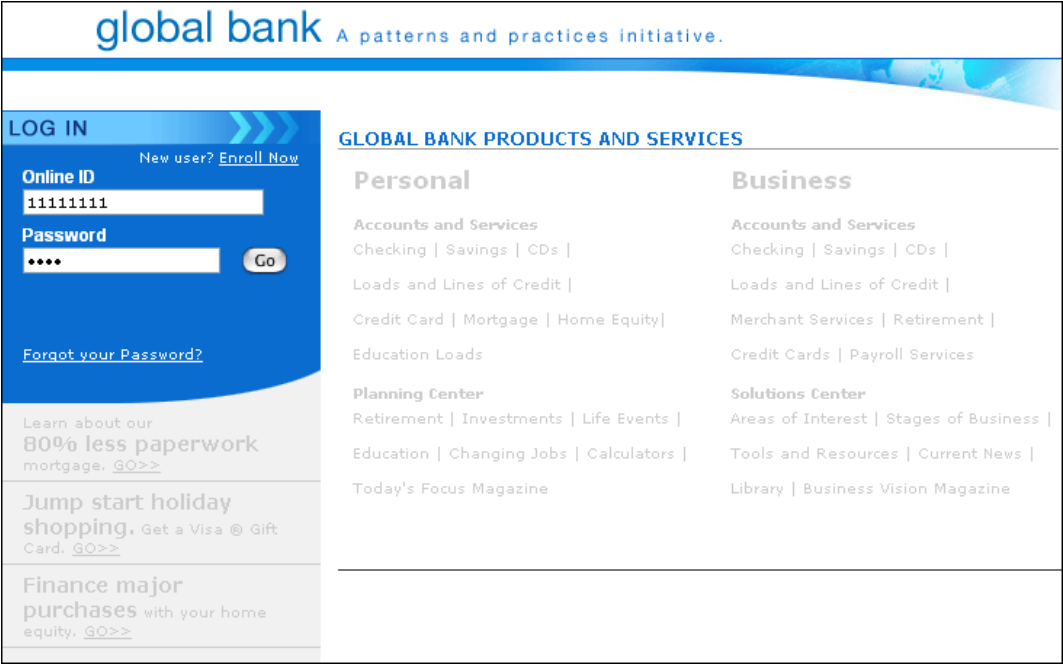


Figure 3
Login page

After the user's credentials are validated, the Account Summary page displays. This page is similar to the page in Figure 4; it shows the user's aggregated summary information. The user can perform tasks such as bill payment, funds transfer, and bill subscription using the menu bar at the top of the page.

global bank A patterns and practices initiative.

Home	Account Summary Transaction Log	Funds Transfer	Bill Subscription Bill Payment	Help
------	------------------------------------	----------------	-----------------------------------	------

CASH ACCOUNTS

Account Type	Account number	Balance(USD)
Checking Account	99-123-1000	9,720.00
Savings Account	99-124-1000	10,015.00

INVESTMENT FUNDS

Name	Shares	Quote
High Risk Fund	1,000.00	12.01
Medium Risk Fund	2,000.00	25.00

CUMULATIVE DEPOSITS

Identifier	Principal	Interest	Term	Action On Due Date	Due Date
123-100-1000	1,000.00	600.00	120	Renew	9/1/2005
123-100-1001	2,000.00	900.00	90	Encash	6/1/2005

CREDIT CARDS

Card Number	Card Type	Balance	Credit Limit	Expiration	Payment Due	Name on
-------------	-----------	---------	--------------	------------	-------------	---------

Figure 4

Account Summary page

The sequence diagram in Figure 5 shows how the user, systems, and services interact to implement the Login and Consolidated Account Sequence use cases.

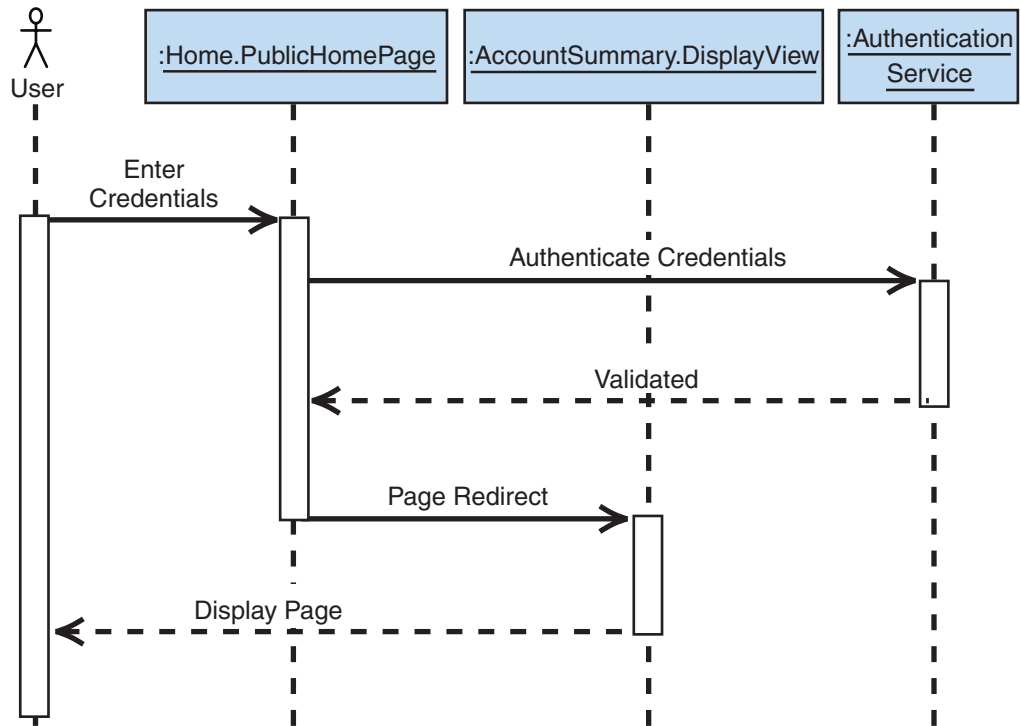


Figure 5

Login and Consolidated Account sequence diagram

Transfer Funds Use Case

Figure 6 shows how user can select the accounts to transfer funds, the amount to transfer, and enter comments.

The screenshot shows a web application interface for a bank. At the top, the logo "global bank" is displayed in blue, followed by the tagline "A patterns and practices initiative." in a smaller, lighter blue font. Below the logo is a horizontal navigation bar with five tabs: "Home", "Account Summary Transaction Log", "Funds Transfer", "Bill Subscription Bill Payment", and "Help". The "Funds Transfer" tab is currently selected. Below the navigation bar, the page title "FUNDS TRANSFER - PREPARATION" is shown in blue. The main form area contains several input fields: "From Account" with a dropdown menu showing "Savings Account 99-124-1000 10,015.00 USD"; "To Account" with a dropdown menu showing "Checking Account 99-123-1000 9,720.00 USD"; "Effective Date" with a text input showing "5/27/2004" and a calendar icon; "Amount (USD)" with a text input showing "800.00"; and "Comments" with a text input showing "Transfer \$800.00 from Savings to Checking account". A note "(* Not implemented)" is located below the "Effective Date" field. A "Prepare" button is located at the bottom right of the form.

global bank A patterns and practices initiative.

Home Account Summary Transaction Log Funds Transfer Bill Subscription Bill Payment Help

FUNDS TRANSFER - PREPARATION

From Account
Savings Account 99-124-1000 10,015.00 USD

To Account
Checking Account 99-123-1000 9,720.00 USD

Effective Date 5/27/2004 Amount (USD) 800.00
(* Not implemented)

Comments
Transfer \$800.00 from Savings to Checking account

Prepare

Figure 6

Prepare page

After the user clicks the **Prepare** button, the Confirm page shown in Figure 7 displays.

global bank A patterns and practices initiative.

Home	Account Summary Transaction Log	Funds Transfer	Bill Subscription Bill Payment	Help
------	------------------------------------	----------------	-----------------------------------	------

FUNDS TRANSFER - CONFIRMATION

Please confirm the following Funds Transfer transaction

From Account	99-124-1000
New Balance	9,215.00
To Account	99-123-1000
Amount (USD)	800.00
Effective Date	5/27/2004
Comments	Transfer \$800.00 from Savings to Checking account

Figure 7

Confirm page

The user clicks the **Confirm** button to complete the funds transfer. The Receipt page in Figure 8 is then displayed. If the user wants to go back to the Prepare page to edit the funds transfer information, the user can click the **Edit** button.

global bank A patterns and practices initiative.

Home	Account Summary Transaction Log	Funds Transfer	Bill Subscription Bill Payment	Help
------	------------------------------------	----------------	-----------------------------------	------

FUNDS TRANSFER - RECEIPT

Your transaction was completed. Use the following number for your future reference: 6

Figure 8

Receipt page

The Receipt page displays the funds transfer transaction reference code. The user clicks the **Done** button to complete the use case.

The sequence diagram in Figure 9 describes how the user, systems, and services interact to implement this use case.

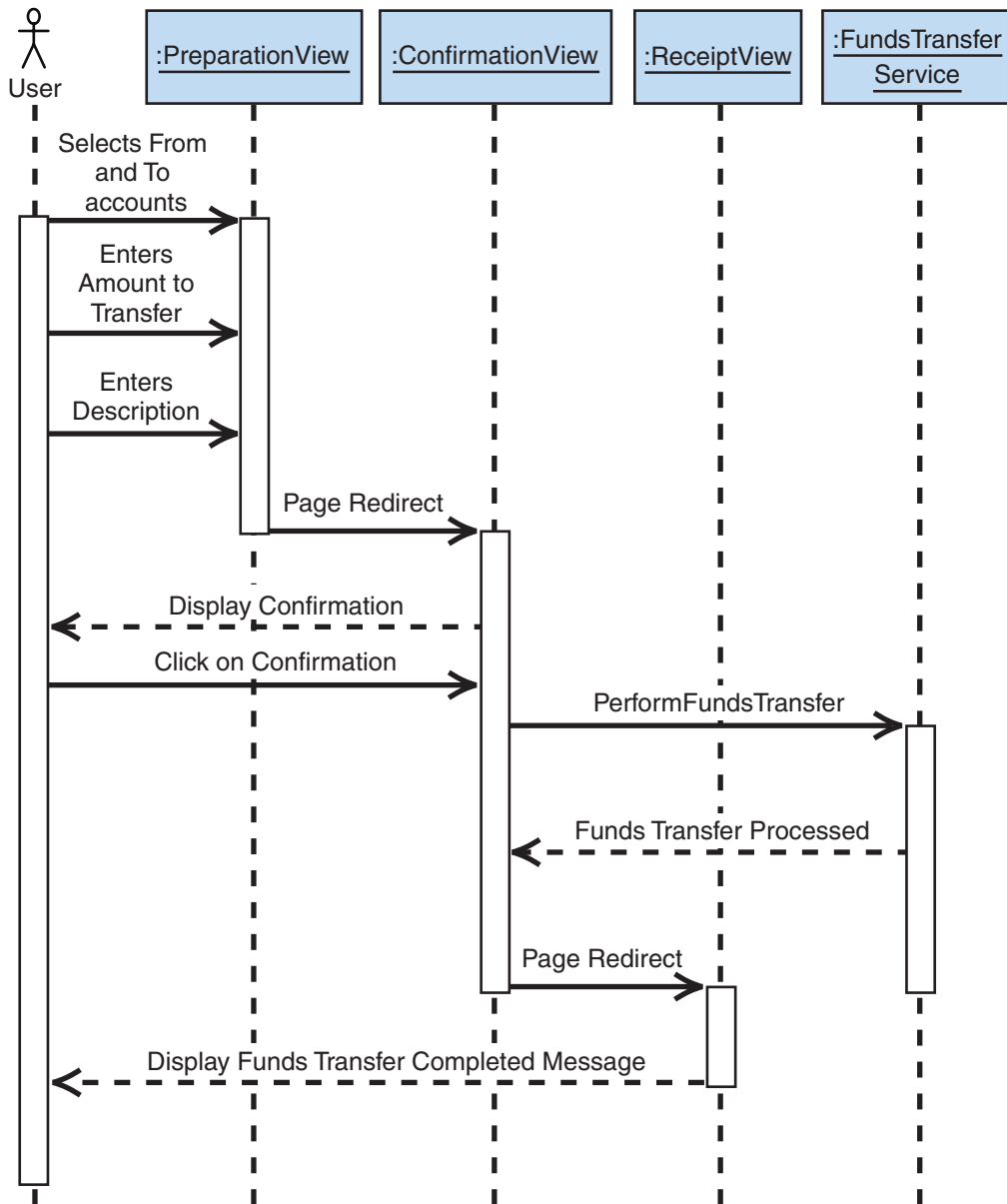


Figure 9

Funds Transfer sequence diagram

Bill Subscriptions Use Case

The Bill Subscriptions use case supports two different actions: adding a bill subscription and deleting a bill subscription.

Add Bill Subscription

In this scenario, the user selects the payee from the drop-down list and enters the payee account number in the text box as shown in Figure 10.

global bank

A patterns and practices initiative.

Home

Account Summary
Transaction Log

Funds Transfer

Bill Subscription
Bill Payment

Help

BILL SUBSCRIPTION

Bill Payee

Global Wireless

Account Number

999702

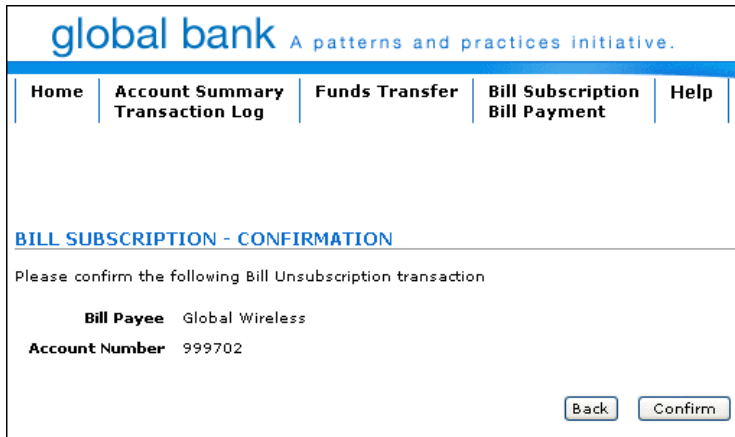
Add

SUBSCRIBED

Bill Payee	Account Number	
Global Wireless	999000	Delete
Global Water Utility	999001	Delete
Global Electricity Utility	999002	Delete

Figure 10
Prepare page

The user clicks the **Add** button to create an additional bill subscription. The Confirm page will summarize the request; this page is shown in Figure 11.



global bank A patterns and practices initiative.

Home Account Summary Transaction Log Funds Transfer Bill Subscription Bill Payment Help

BILL SUBSCRIPTION - CONFIRMATION

Please confirm the following Bill Unsubscription transaction

Bill Payee Global Wireless

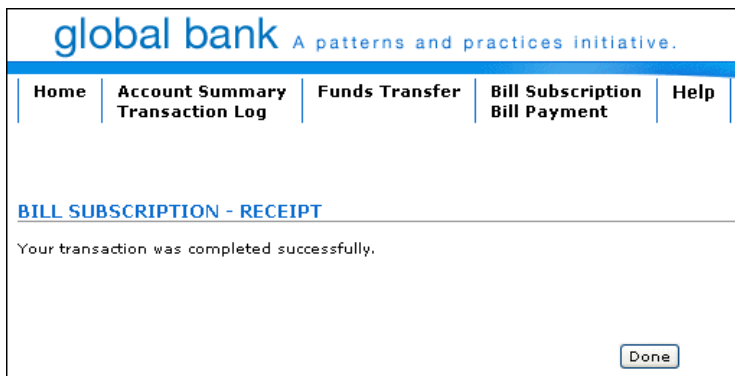
Account Number 999702

Back Confirm

Figure 11

Confirm page

The user clicks the **Confirm** button to complete the bill subscription addition, and the Receipt page is displayed; Figure 12 shows this page. The user can also click the **Back** button to go back to the Prepare page to edit the bill subscription information.



global bank A patterns and practices initiative.

Home Account Summary Transaction Log Funds Transfer Bill Subscription Bill Payment Help

BILL SUBSCRIPTION - RECEIPT

Your transaction was completed successfully.

Done

Figure 12

Receipt page

The Receipt page displays a confirmation message that the task is complete. The user clicks the **Done** button to complete the use case.

The sequence diagram in Figure 13 describes the steps used to add a bill subscription.

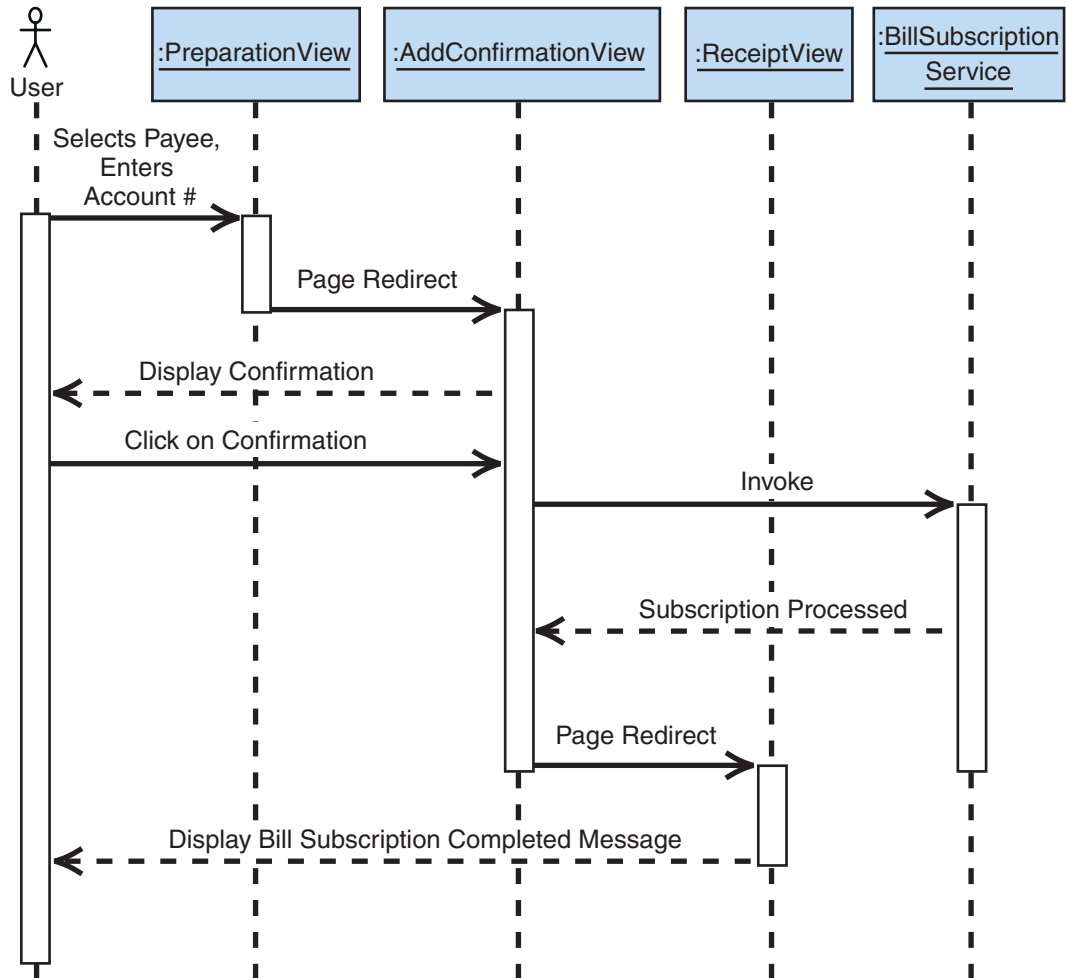
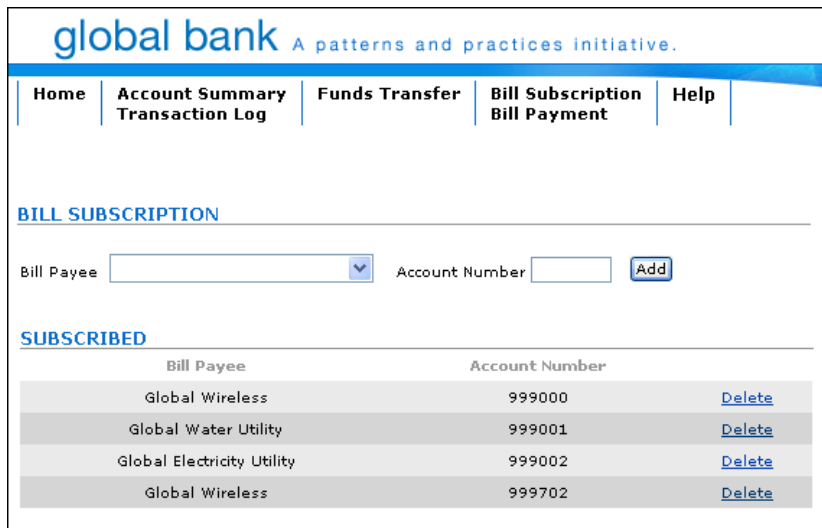


Figure 13

Add Bill Subscription sequence diagram

Delete Bill Subscription

The user deletes a bill subscription by clicking on the **Delete** link next to the bill in the **Subscribed** section as shown in Figure 14.



global bank A patterns and practices initiative.

Home Account Summary Transaction Log Funds Transfer Bill Subscription Bill Payment Help

BILL SUBSCRIPTION

Bill Payee Account Number


SUBSCRIBED

Bill Payee	Account Number	
Global Wireless	999000	Delete
Global Water Utility	999001	Delete
Global Electricity Utility	999002	Delete
Global Wireless	999702	Delete

Figure 14

Prepare page

The Confirm page, as shown in Figure 15, displays.



global bank A patterns and practices initiative.

Home Account Summary Transaction Log Funds Transfer Bill Subscription Bill Payment Help

BILL SUBSCRIPTION - CONFIRMATION

Please confirm the following Bill Subscription transaction

Bill Payee Global Wireless

Account Number 999702

Figure 15

Confirm page

The user clicks the **Confirm** button to complete the bill subscription deletion, and the Receipt page displays; this page is shown in Figure 16. The user can also click the **Back** button to go back to the Prepare page to edit the bill subscription.

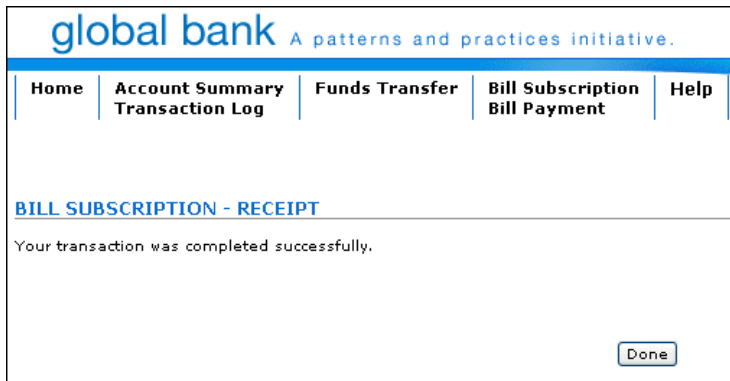


Figure 16

Receipt page

The Receipt page displays a confirmation message that the task is complete. The user clicks the **Done** button to complete the use case.

The sequence diagram in Figure 17 describes the steps used to delete a bill subscription.

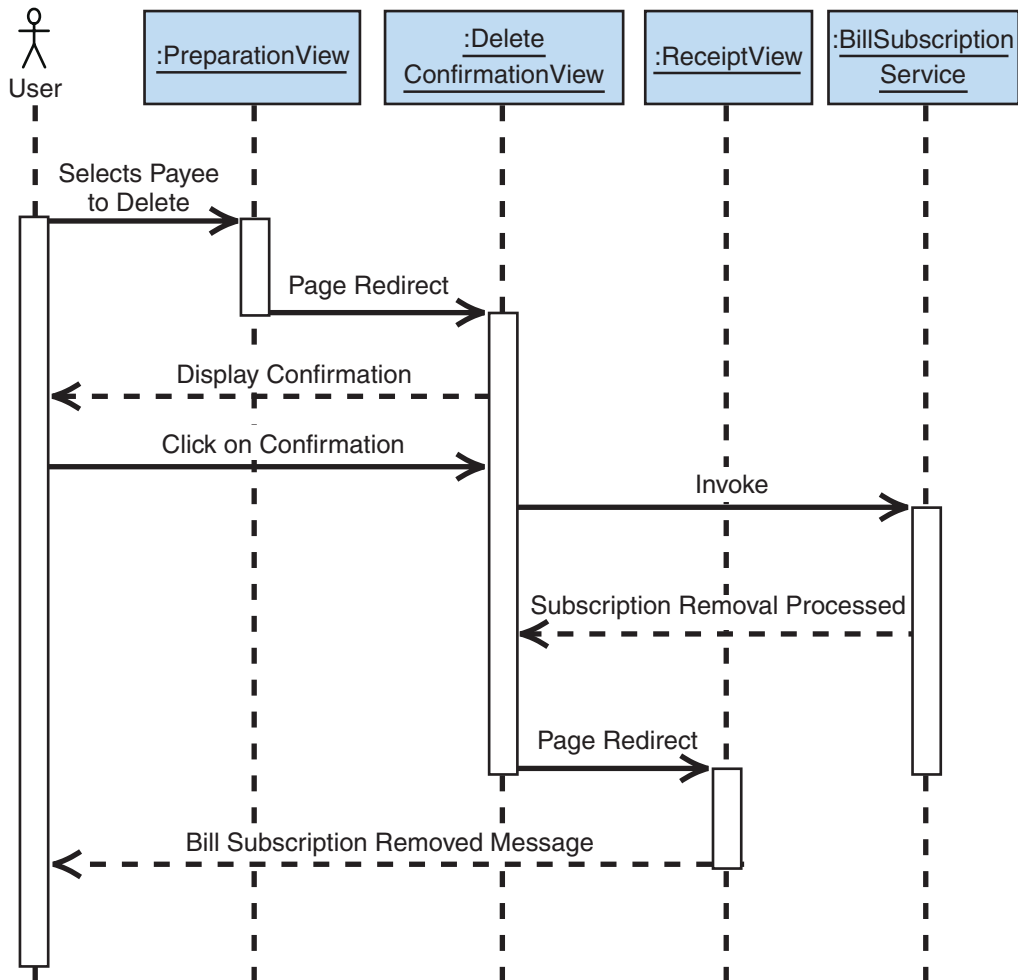


Figure 17

Delete Bill Subscription sequence diagram

Bill Payment Use Case

In this use case, the user pays a bill. The user first selects the source account used to pay the bill from the drop-down list. The pending bills in the grid are listed as shown in Figure 18.

global bank

A patterns and practices initiative.

Home

Account Summary
Transaction Log

Funds Transfer

Bill Subscription
Bill Payment

Help

BILL PAYMENT

From Account

Checking Account 99-123-1000 10,520.00 USD

PENDING BILLS

Bill Payee	Account Number	Due Date	Amount (USD)	
Global Wireless	BILL_1_100007	6/30/2004	100.00	Select
Global Water Utility	BILL_1_100001	7/30/2004	50.00	Select
Global Water Utility	BILL_1_100004	7/15/2004	75.00	Select
Global Electricity Utility	BILL_1_100002	6/30/2004	100.00	Select
Global Electricity Utility	BILL_1_100005	8/1/2004	100.00	Select

Prepare

Figure 18
Prepare Account page

The user selects the bill to pay by clicking the **Select** link next to the appropriate pending bill, as shown in Figure 19.

global bank A patterns and practices initiative.

Home | Account Summary Transaction Log | Funds Transfer | Bill Subscription Bill Payment | Help

BILL PAYMENT

From Account
 Checking Account 99-123-1000 10,520.00 USD ▼

PENDING BILLS

Bill Payee	Account Number	Due Date	Amount (USD)	
Global Wireless	BILL_1_100007	6/30/2004	100.00	Select
Global Water Utility	BILL_1_100001	7/30/2004	50.00	Select
Global Water Utility	BILL_1_100004	7/15/2004	75.00	Select
Global Electricity Utility	BILL_1_100002	6/30/2004	100.00	Select
Global Electricity Utility	BILL_1_100005	8/1/2004	100.00	Select

Prepare

Figure 19

Prepare Bill page

The Confirm page, as shown in Figure 20, displays.

global bank A patterns and practices initiative.

Home | Account Summary Transaction Log | Funds Transfer | Bill Subscription Bill Payment | Help

BILL PAYMENT - CONFIRMATION

Please confirm the following Bill Payment transaction

From Account 99-123-1000
New Balance 10,420.00
Bill Payee Global Wireless
Account Number BILL_1_100007
Amount (USD) 100.00

Back Confirm

Figure 20

Confirm page

The user clicks the **Confirm** button to complete the bill payment, and the Receipt page displays; this page is shown in Figure 21. The user can also click the **Back** button to go back to the Prepare page to edit the bill payment information.

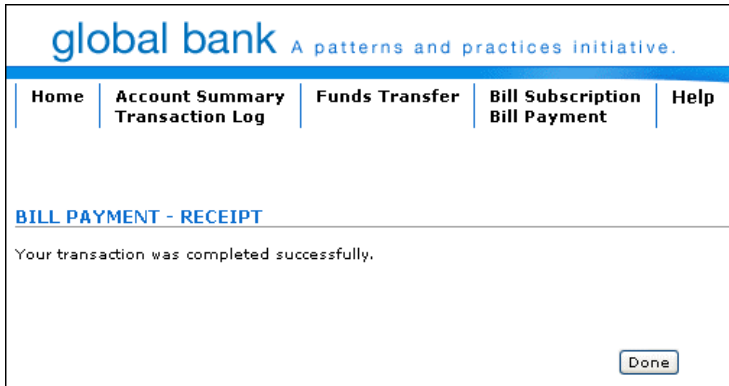


Figure 21

Receipt page

The Receipt page displays a confirmation message that the task is complete. The user clicks the **Done** button to complete the use case.

The sequence diagram in Figure 22 describes the steps used to pay bill subscriptions.

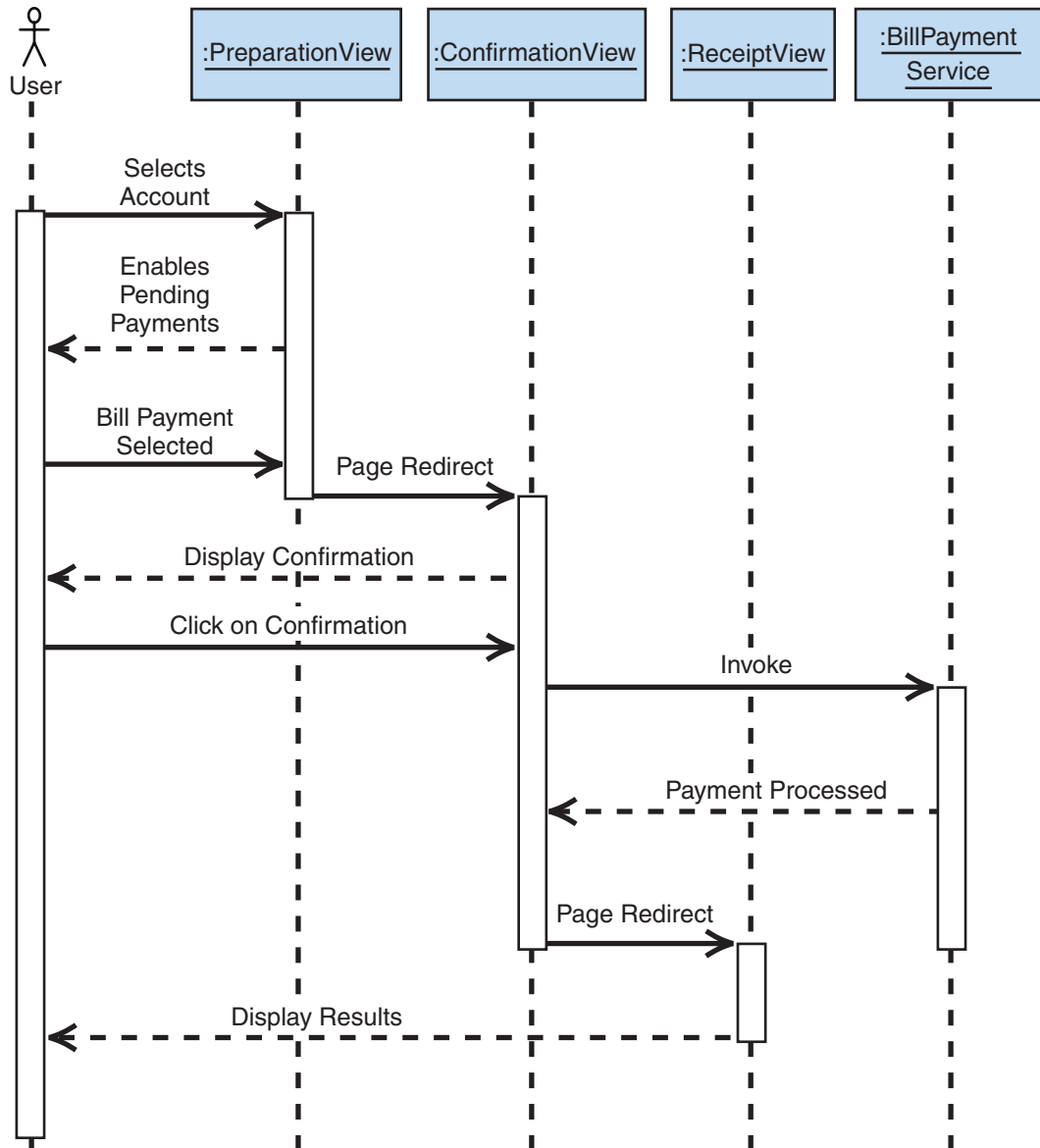


Figure 22

Bill Payment sequence diagram

Transaction Log Report Use Case

When the user views this page, the default transaction log data for the current date displays. The user can alter the **From** and **To** dates and click the **Get Transaction Log** button to request the transaction log data for the specified date range, as shown in Figure 23.

global bank

A patterns and practices initiative.

Home

Account Summary
Transaction Log

Funds Transfer

Bill Subscription
Bill Payment

Help

From

5/27/2004

To

5/27/2004

Get Transaction Log

Transaction Log from date:5/27/2004 to date:5/27/2004.

FUNDS TRANSFER

From Account	To Account	Amount (USD)	Effective Date	Comments
99-124-1000	99-123-1000	785.00	5/27/2004	Transfer \$785.00 from Savings to Checking account
99-123-1000	99-124-1000	900.00	5/27/2004	Transfer \$900.00
99-124-1000	99-123-1000	800.00	5/27/2004	Transfer \$800.00 from Savings to Checking account

BILLS

Bill Payee	Account Number	From Account	Amount (USD)	Effective Date
Global Wireless	BILL_1_100000	99-123-1000	100.00	5/27/2004
Global Wireless	BILL_1_100003	99-123-1000	65.00	5/27/2004
Global Wireless	BILL_1_100006	99-124-1000	100.00	5/27/2004
Global Wireless	BILL_1_100007	99-123-1000	100.00	5/27/2004

Figure 23

Transaction Log page

The sequence diagram in Figure 24 describes how the user, systems, and services interact to implement this use case.

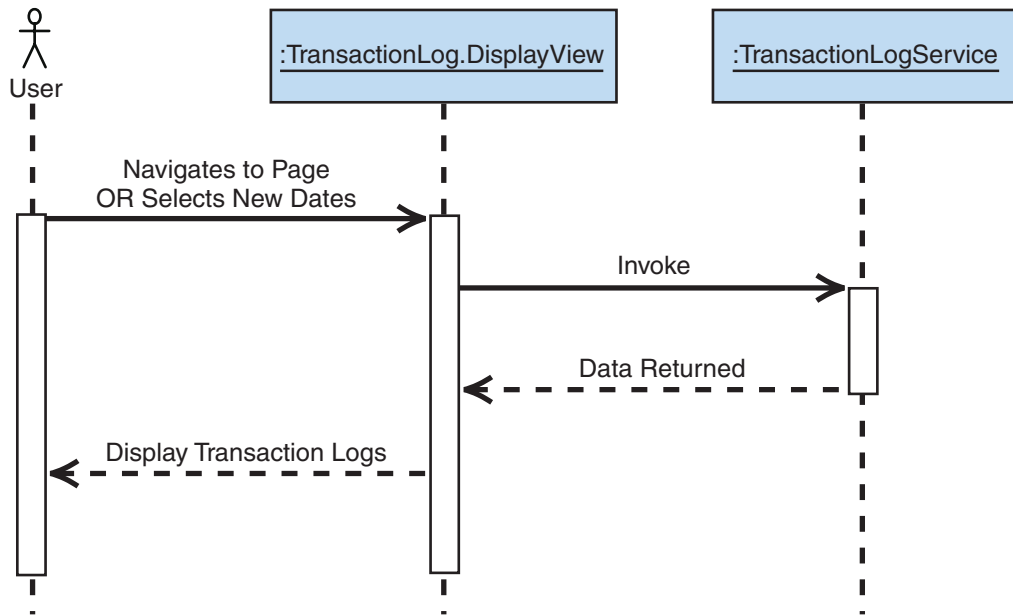


Figure 24

Transaction Log sequence diagram

2

Architecture

Good software architecture provides both a blueprint of a system (a detailed design) and an abstraction that serves to manage the complexities of the design so that the application's intent and functionality can be more easily understood.*

This section describes the major structural elements of the EDRI. It explains and illustrates the architecture from a number of perspectives. Collectively, these representations provide a comprehensive and detailed introduction to the reference implementation, how it is designed to function, and how it is organized.

Design Objectives and Principles

Global Bank is an established national bank that serves customers and businesses that are small to medium in size. Global Bank has made a series of strategic acquisitions in recent years to help sustain growth in emerging markets. Global Bank plans to develop an online banking portal to allow customers to access personal finance information online.

Like many large organizations, Global Bank has a diverse range of back-end technologies that support the day-to-day operations of the bank. Many of these technologies run on proprietary software and hardware with limited capability for reuse. Global Bank is committed to extending its functional integration strategy to incorporate standards-based Web services — as a move toward broader adoption of a service oriented architecture.

* Hofmeister, Christine, Robert Nord, and Dilip Soni. *Applied Software Architecture*. Reading, Massachusetts: Addison-Wesley, 1999

Objectives

In addition to meeting the business requirements for an online banking application, the Global Bank chief architect responsible for the enterprise architecture has stated two strategic objectives:

- Standardize using an enterprise framework to ensure consistent and timely development of business logic.
- Establish standards for using service-oriented integration as a means to expose system functionality so that other channels, such as Teller Terminals, Voice Response Systems, and ATMs, can use the same functionality at a later point.

Enterprise Framework

After studying the options, the project architect chose to standardize development of services using the *patterns & practices* Enterprise Development Reference Architecture. This solution will facilitate the following:

- Separation of business logic from underlying transports.
- Separation of cross-cutting concerns from business logic.
- Separation of Service Interface from Service Implementation to support resiliency and alternate deployment scenarios for services.

To understand how the framework processes requests and associated details, see “Appendix A — Inside the Enterprise Development Application Framework.” The following sections discuss the architecture of the Global Bank online banking system.

Service Oriented Integration

Over the years, Global Bank invested in technologies that were not always designed with interoperability in mind. Like most organizations, Global Bank does not have the luxury of rewriting these applications when new technologies emerge.

The project architect is following the guidance contained within the *patterns & practices* Functional Integration pattern to integrate information systems that were not designed to work together. The service-oriented integration approach to functional integration will enable system functionality to be exposed using standards-based Web services.

The project architect proposed a small number of Web services that will provide access to disparate back-end systems. Each service was designed to do the following:

- Generalize functionality independently of platform or application specific execution context.
- Use an XML Schema to specify a service's structure. This reduces type system coupling and allows for automated validation of a message against a schema.
- Protect the integrity of each system's data by validating all requests to a service. This includes ensuring only trusted clients can call the service.
- Plan for unavailability of dependent services within a client. For example, if the Investment Fund System is unavailable, the Account Statement Service should still return data from the other systems that are available.
- Separate enforcement of a service's policy requirements, such as authentication, from business logic to simplify adoption of emerging WS-Policy specifications in the future.

Architectural Representation

A software architecture is a complex, multifaceted set of artifacts that cannot be fully explained in a single diagram or from a single viewpoint. To help explain these artifacts, software architects typically use a number of different perspectives, or *views*, to depict architectures. In this chapter, the combination of these views provides a comprehensive picture of the EDRI, its functional elements, and the interactions between those elements. The views and their objectives are as follows:

- **Conceptual view.** The conceptual view is a high-level overview of the key architecture elements and their relationships.
- **Logical view.** The logical view is a detailed description of key elements of the architecture. The view describes the grouping of design elements (classes and interfaces) into packages represented as namespaces; it also describes the static and dynamic relationships between the classes.
- **Implementation view.** The implementation view describes how the classes and interfaces are organized into directories, projects, and assemblies in the file system and in the Microsoft® Visual Studio® development system.
- **Deployment view.** For the system architecture, this view documents the likely physical topology. It includes each computer in the implementation and describes how they are interconnected. The configuration for each node is also specified — operating system, database, and applications.

For the services, the deployment view shows the distribution of components across distinct processing nodes.

Terminology and Key Concepts

Before you begin to review the architectural details of the EDRI, it would be helpful to review the following terms. Many of these terms are commonly used within the industry; however, some of these terms may have different connotations in this document or in the field of software architecture in general. (Note that these terms are presented in logical order, rather than the more typical alphabetical sequencing of a glossary.)

Reference architecture

Architectural blueprint describing a selection and composition of architectural patterns best addressing the needs of a class of solutions with known properties.

Application framework

A domain-specific, partially complete software (sub-) system that is intended to be instantiated. Incorporates provision for extensions and adaptations.

Reference implementation

Partial implementation of a solution for a compelling business scenario. Implements selected use cases in the scenario. Developed with the explicit objective of demonstrating the use of a reference architecture exemplifying prescriptive guidance.

Pattern

A description of a recurring problem that occurs in a given context and, based on a set of guiding forces, recommends a solution. The solution is usually a simple mechanism: a collaboration between two or more classes, objects, services, processes, threads, components, or nodes that work together to resolve the problem identified in the pattern.

Cross-cutting concern

A type of functionality that can be applied to multiple classes and/or applications as they do not typically relate to a specific business problem. Typical cross-cutting concerns include functionality such as authentication, authorization, and application instrumentation.

Loosely coupled

A type of distributed application that is designed to function autonomously. Design and implementation make few assumptions about the application in which they interact, and can be deployed and versioned independently.

Conceptual View

The conceptual view identifies, at a high level, the interaction of architectural elements. It serves to identify the project vision based on previously identified business and user requirements.

Dominant Patterns

The process design requirements for the Global Bank Internet banking application center around three considerations:

- **Separation of viewer and controller logic.** This involves ensuring that the logic of rendering the presentation is separated from the logic for interacting with the model (services).
- **Navigation between views of a use case.** This ensures that the logic is centralized into classes for maintainability and consistency.
- **Managing state when navigating between views.** This preserves state across the views connected to a particular use case. It provides the views of the use case with consistent access to the data.

To separate the user interface implementation from the navigation logic, the Global Bank Internet banking application implements two patterns; the Model-View-Controller pattern and a variant of the Model-View-Controller pattern — the Page Controller pattern. The Presentation tier uses variants of the Model-Viewer-Controller pattern. The Page Controller pattern is intended for Web applications where navigation is fixed and complexity is relatively small. The result is a Web-based application that is easy to develop and extend as new functional areas are added to the system.

Model-View-Controller Pattern

The traditional Model-View-Controller pattern was used to implement the following three use cases:

- Login
- Consolidated Account Statement
- Transaction Log Report

In this implementation, the views are the ASP.NET Web pages. The controllers are separate classes used by these views. And the model is implemented within the Web service. In these use cases, the controller and view are independent, and the view has a reference to the instance of the controller. The controller logic can be invoked by several views but only by the views within the appropriate use case. For these use cases, the controller can be used by views in any of the use cases. For more information, see the [Model-View-Controller pattern](#) in *Enterprise Solution Patterns Using Microsoft .NET*, Redmond: Microsoft Press, 2003.

Page Controller Pattern

For the other more complex use cases, the Page Controller pattern was used to more clearly separate the view from the controller. Because the controller and view derive from the same class without additional functionality, the controller can access all functionality the **Page** class provides. In addition, the state was shared and could easily be modified.

This pattern implies that the views derive from the controllers. There is an inherent dependency between views and controller due to this relationship, and therefore, some of the state manipulation logic can be found in the views. All of the interactions with the model, which in this case is the services, is found only in the controller, therefore preserving a clean separation of responsibilities. It is important to keep in mind the complexity of the UI design when using this pattern. For more information, see the [Page Controller pattern](#) in *Enterprise Solution Patterns Using Microsoft .NET*.

In the Global Bank application, this pattern is applied with a small variation: There is one controller for each use case. This can be seen in the Bill Payment, Bill Subscription, and Funds Transfer use cases. The Web application design used here is targeted at Web applications with a relatively simple UI requirement where each use case typically is associated with around three screens.

Logical View

The logical view presents the core design of the system. It presents the primary classes that collaborate to implement the system functionality and communicate their behavior. With respect to this Web application, the focus is on the pages that provide the core set of functionality and the page controller classes they derive from.

Model-View-Controller Pattern

As mentioned above, the Model-View-Controller pattern is used in the Login, Account Summary, and Transaction Log use cases. This pattern uses a separate class for the view, the controller, and the model as shown in Figure 1. They are aligned according to the use cases. In this Web application, the pattern is applied to the following functional areas:

- Login
- Account Summary
- Transaction Log

Each Web page (view) employs a controller class. The controller class uses a Web service proxy to invoke the desired service. The only exception is that the view for the authentication functional area is an ASP.NET user control rather than a Web page. The following class diagram illustrates the relationships between these classes:

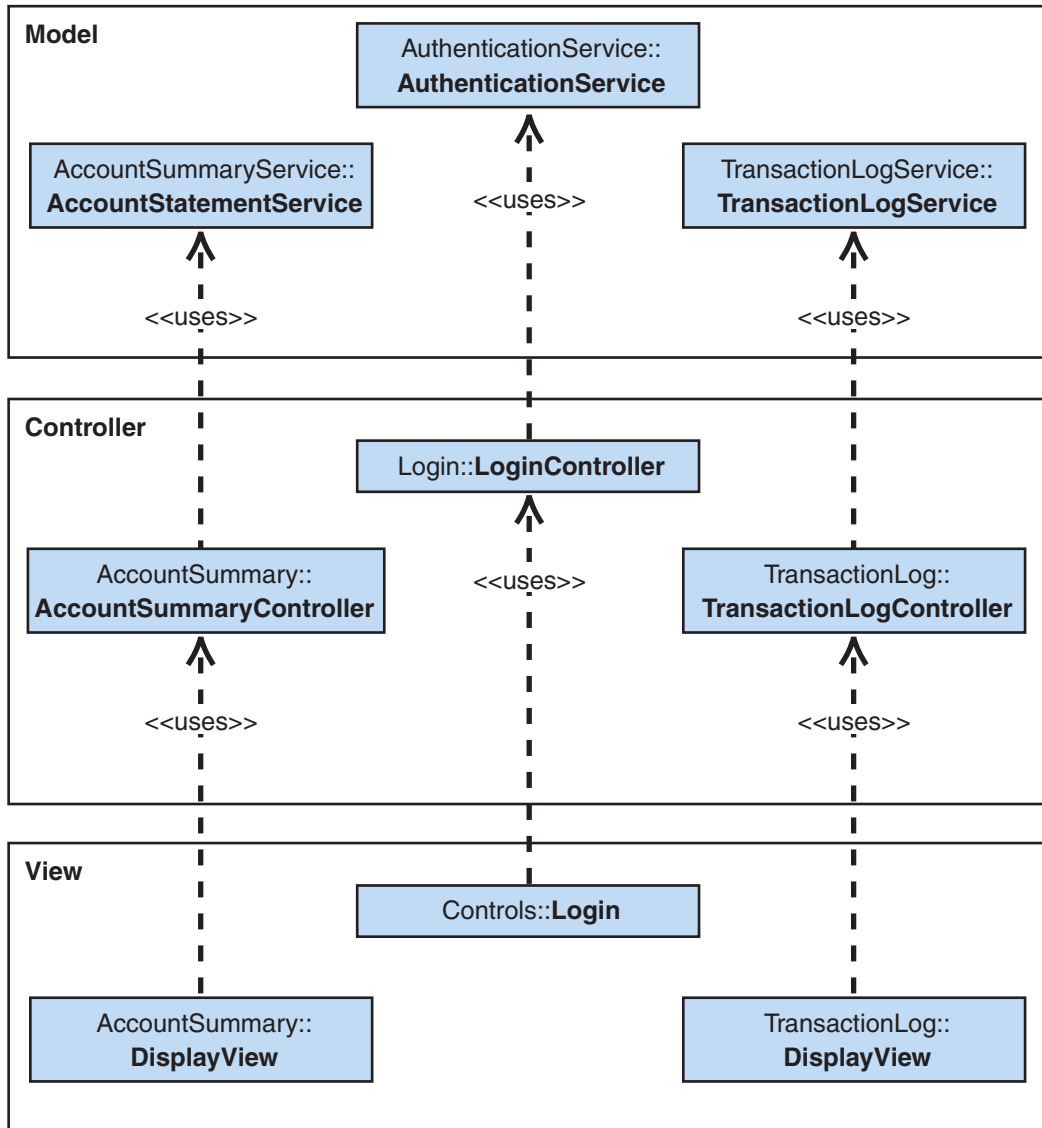


Figure 1
Model-View-Controller pattern

Page Controller Pattern

The classes involved in the implementation of the Page Controller Pattern in the Global Bank Internet banking application are the following:

- **View (Web page).** This is implemented by the .aspx pages in the application. They are responsible for rendering the data for presentation. For example, the Bill Payment use case views: Bill\Payment\Confirmation.aspx, Bill\Payment\Preparation.aspx, Bill\Subscription\Receipt.aspx.
- **Controller.** There is one controller for each use case. This class is responsible for navigation between the views of the use case, preserving the use case state during view navigation, and communicating with the Web services. The two billing uses cases have the following page controllers: Bill\Payment\PageController.cs, Bill\Subscription\PageController.cs.
- **Navigation Controller.** All controllers derive from this class. This class provides the functionality for navigating between the views and preserving the use case state. There is only one **PageNavigationController** class.
- **Graph.** There is one graph (navigation graph) per use case. This class is responsible for managing the view navigation graph. It contains information about which are valid views to navigate to and from for any given view. You can see a **graph** class in Bill\Payment\Graph.cs or Bill\Subscription\Graph.cs.
- **Webview.** This represents a Web page view. It contains the data specific to a Web page view, namely a URL; the valid navigable views from this page in the form of a navigation graph object; and the logic to determine whether a specified view is valid to navigate to. There is only one **webview** class.
- **Use case state.** This class is used to preserve the state for the use case. The state contains the user entered information and can be used in other views of the use case. This is stored in a user-defined **struct** type. There is one state type for each use case. The state is preserved during the use case in the session state. Two examples are Bill\Payment\BillPaymentState.cs and Bill\Subscription\BillSubscriptionState.cs.

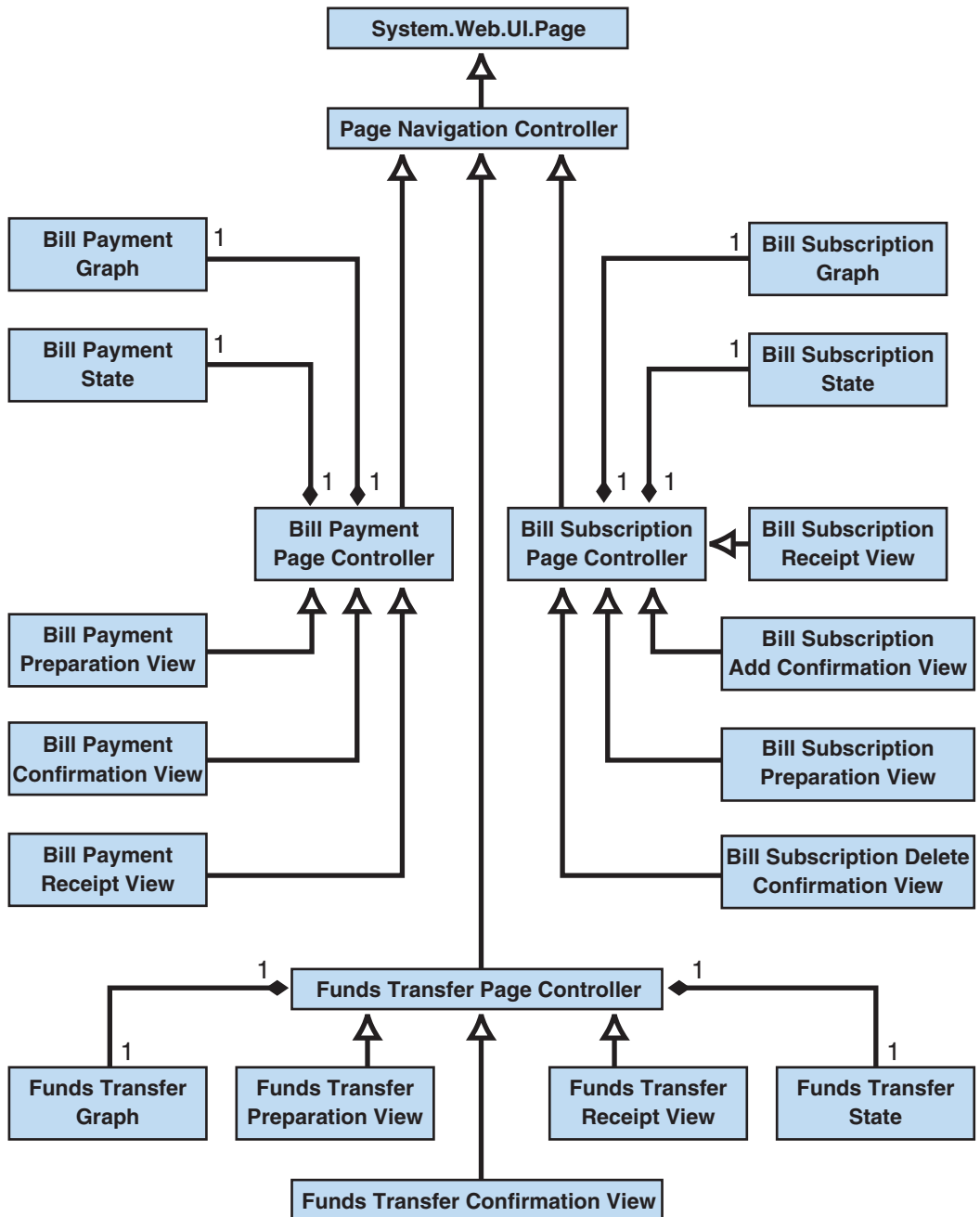
The Web application uses the Page Controller pattern because the user interface is relatively simple and does not warrant the additional complexity of the Front Controller pattern or the UIP Application Block. For more information, see the [Front Controller pattern](#) in *Enterprise Solution Patterns Using Microsoft .NET*.

You should consider using the UIP Application Block where you have complex decision paths, and where you want to have configurable navigation graphs. To qualify, each use case should have at least five screens. To determine whether it fits your requirements, see the [User Interface Process \(UIP\) Application Block – Version 2.0](#).

The Page Controller pattern uses a page controller class for each functional area. This page controller class derives from a single page navigation controller class. In this Web application, there are only three dynamic functional areas:

- Bill payment
- Bill subscription
- Funds transfer

Each Web page belonging to a functional area derives from that area's page controller class. For example, the preparation, confirmation, and receipt views for the funds transfer functionality area all derive from the funds transfer page controller class. The class diagram in Figure 2 illustrates the relationships between these classes.

**Figure 2**

Implementation of Page Controller pattern in the EDRI

Funds Transfer Walkthrough

Using the Funds Transfer use case as a scenario, Figure 3 describes how these objects collaborate in the Page Controller pattern to perform a funds transfer from one account to another.

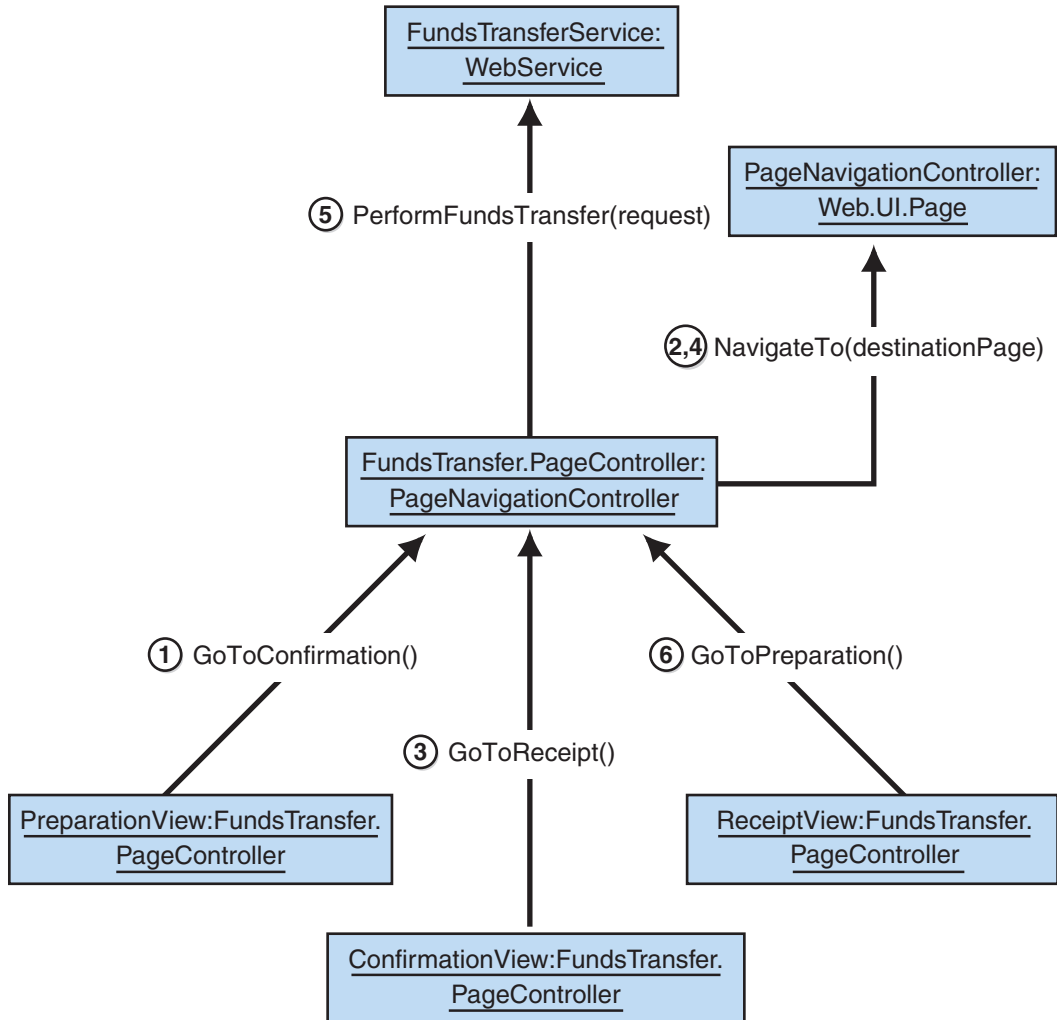


Figure 3

Funds Transfer walkthrough

This walkthrough begins with the user clicking on the funds transfer link of the navigation bar at the top of a Web page. This loads the `Preparation.aspx` page. The walkthrough includes the following steps:

1. After the user enters the appropriate information for a funds transfer, he or she clicks the **Prepare** button. This calls the **GotoConfirmation()** method of the **PageController** class. The page controller class now loads the `Confirmation.aspx` page using the **Server.Transfer()** method.
2. After reviewing the information entered on the preparation page, the user clicks the **Confirm** button (as opposed to returning to the preparation page by clicking the **Edit** button). This calls the **GotoReceipt()** method of the **FundsTransferPageController** class.
3. The **GotoReceipt()** method performs the service request, invoking the **PerfromFundsTransfer()** method of the **FundsTransferService** class. It passes the method an instance of the **FundsTransferRequest** class. When the service completes, a response is returned to the page controller class. The controller class now loads the `Receipt.aspx` page.
4. The `Receipt.aspx` page uses the response to display the results of the operation. When the user clicks the **Done** button, the **GotoPreparation()** method of the **FundsTransferPageController** class is invoked. The page controller class now loads the `Preparation.aspx` page using the **Server.Transfer()** method.

Service Invocation

The Global Bank Internet banking application invokes the services it uses through the use of Web service proxies. These proxies were compiled using the .NET WSDL utility by passing in the service URL. This tool generates a class in the specified language, Microsoft Visual C#[®] development tool in this case, that maps the parameters for a Web method to XML elements in a SOAP message. For more information about generating a Web service proxy with the WSDL utility, see [Creating the Web Service Proxy](#) on MSDN.

For the Global Bank application, Web service proxies were generated for each of the services used: **AuthenticationService**, **AccountStatementService**, **BillPaymentService**, **BillSubscriptionService**, **FundsTransferService** and **TransactionLogService**. The WSDL utility creates these classes so that they derive from the **SoapHttpClientProtocol** class found in the **System.Web.Services.Protocols** namespace. Each Web service proxy class was added to the Global Bank solution. They can be found in the **WebApplication.WebServiceProxies** assembly.

Returning to the Funds Transfer use case scenario to demonstrate how this works, the **PageController** class for the funds transfer functional area calls the service proxy from within the **GotoReceipt()** method. The following code examples illustrate how to perform this action in code. First, an instance of the proxy is created.

```
// create an instance of the service proxy
FundsTransferService service = new FundsTransferService();
```

Next, an instance of the service's request header is created and populated with the user's login ID using the following code.

```
RequestHeader requestHeaders = new RequestHeader();
requestHeaders.Headers = new object[2];
object[] userName = new object[2];
userName[0] = "UserName";
userName[1] = "11111111";
object[] channel = new object[2];
channel[0] = "Channel";
channel[1] = "Home Banking";
requestHeaders.Headers[0] = userName;
requestHeaders.Headers[1] = channel;
```

Then an instance of the **FundsTransferRequest** class is created and initialized with the contents of the message.

```
FundsTransferRequest request = new FundsTransferRequest();
// the request parameters are filled in here...
request.SourceAccountNumber = this.fundsTransferState.SourceAccountNumber;
request.DestinationAccountNumber =
this.fundsTransferState.DestinationAccountNumber;
request.Amount = this.fundsTransferState.TransferAmount;
request.EffectiveOn = this.fundsTransferState.TransferEffectiveDate;
request.Description = this.fundsTransferState.Description;
```

Finally, the Web method is invoked using the service proxy, passing it the request object.

```
// invoke the service...
FundsTransferResponse response = service.PerformFundsTransfer(request);
```

The preceding code example demonstrates how to invoke the **PerformFundsTransfer()** Web method using the Web service proxy **FundsTransferService**. When the service proxy is called, it generates a SOAP message and invokes the Web service. At this point, the Enterprise Development Application Framework (EDAF) intercepts the request and begins processing it. Ultimately, the EDAF calls the business action described in the FundsTransfer Service section. When the business action completes, it returns a response. The response sent back through the EDAF, to the service proxy, and then back to the calling function, **GotoReceipt()**, in the page controller class.

The information in the SOAP header includes the channel name, which is currently used only for auditing purposes. In the future, different types of channels may be supported to expose additional capabilities, for instance smart client or interactive voice response channel.

Implementation View

The implementation view maps the classes from the logical view to their physical structure in the solution projects. The implementation view serves as the roadmap from the source code to the physical binaries.

The two main assemblies built for the Global Bank Internet banking application are the **WebApplicationUI** and the **WebApplication.UIProcess**. The first, the **WebApplicationUI**, contains the actual pages viewed by a user. The **WebApplication.UIProcess** assembly contains the page controller classes these pages derive from. Therefore, the **WebApplicationUI** assembly depends on the **WebApplication.UIProcess** assembly. The diagram in Figure 4 illustrates this dependency.

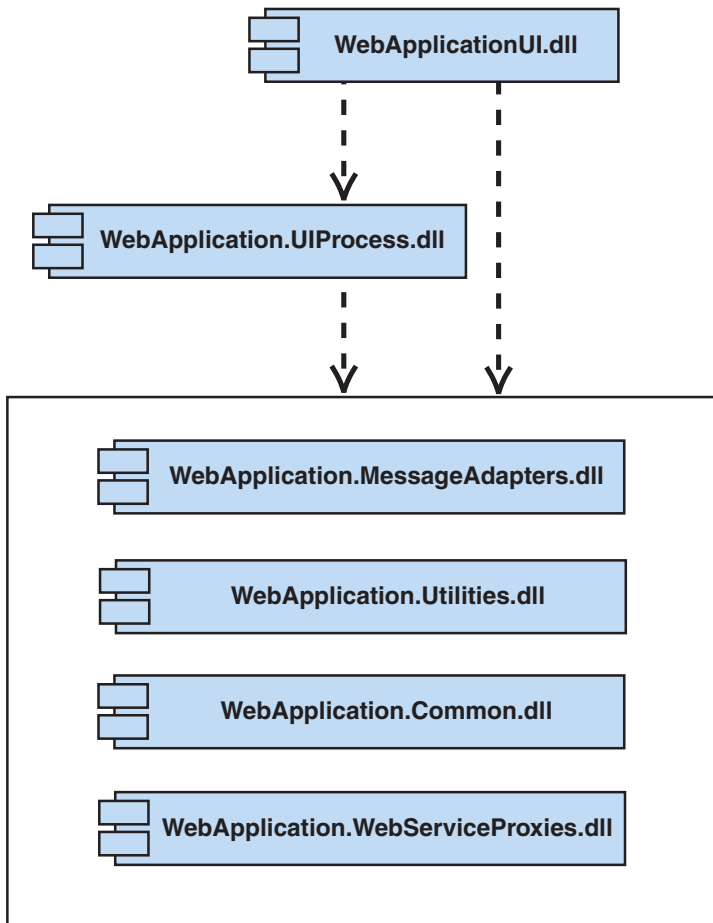


Figure 4

Web application assembly dependencies

The Web application has four additional assemblies that provide some common functionality and the proxies for the Web services. Both the **WebApplicationUI** and the **WebApplication.UIProcess** assemblies reference all four assemblies.

WebApplicationUI

The **WebApplicationUI** assembly contains several subfolders. These subfolders organize the classes into groups and subgroups. The namespaces all begin with **Microsoft.ReferenceImplementation.WebApplication**. For example, the **Display.aspx** class found in the AccountSummary folder within the Task folder has the following namespace:

```
Microsoft.ReferenceImplementation.WebApplication.WebApplicationUI.Task.  
AccountSummary
```

As the following folder structure demonstrates, the assembly contains the Task, Home, and Controls top-level folders. Within the Task folder, there are several subfolders that are used to organize the Web pages by use case functional area.

```
- WebApplicationUI  
  + bin  
  + Controls  
  + Home  
  + mm  
  - Task  
    + AccountSummary  
    - Bill  
      + Payment  
      + Subscription  
    + FundsTransfer  
    + TransactionLog  
  - tools  
    + css  
    + js
```

The Home directory contains the initial pages that are displayed when a user first visits the site. The public page is for anyone visiting the site, while the private page is for authenticated users.

The Controls directory contains several ASP.NET user controls. These are used as custom controls in the application. The **AmountPicker** and **DatePicker** are text boxes that allow users to enter amounts and dates respectively. The **LoggedInHeader** is used to provide navigation. And the **Login** control is used for ASP.NET forms authentication. The user enters their credentials here so they can be validated by the system.

The Task directory contains all of the Web pages (.aspx) and associated classes pertaining to the user interface for each use case. The tools directory contains cascading style sheets (.css) and Java script files. The mm directory contains the images for the user interface. The bin directory contains the binary files.

WebApplication.UIProcess

The same can be done for the **WebApplication.UIProcess** assembly. The directories contain the page controller classes for each functional areas corresponding to the use cases. The **PageController** classes are what the Web pages derive from. The <Installation Location>\Microsoft EDRA\CS\ReferenceImplementation\GlobalBank\WebApplication\UIProcess directory contains the following files and folders.

- UIProcess
 - + bin
 - Controller
 - + AccountSummary
 - Bill
 - + Payment
 - + Subscription
 - + FundsTransfer
 - + Login
 - + TransactionLog

The common files are in the UIProcess directory. The controllers corresponding to each of the use cases are located in each of the subfolders in the Controller directory. The bin directory contains the binary files.

These two assemblies, **WebApplicationUI** and **WebApplication.UIProcess**, make up the majority of the Global Bank Internet banking application. The Page Controller pattern makes it easy to expand this application through new page controllers as new functionality is made available.

Configuration

Global Bank uses two configuration files:

- **GlobalBank.config**. Similar to the ServicesReferenceArchitecture.config file in the EDRA ApplicationTemplate, this file contains the configuration information for the transports, targets, handlers, framework helpers, and events.
- **GlobalBankServices.config**. Similar to the TemplateServices.config file in the EDRA ApplicationTemplate, this file contains the configuration information for Global Bank's Service Interface pipeline, Service Implementation pipeline, and business actions.

These files can be found in <Installation Location>\Microsoft EDRA\CS\ReferenceImplementation\GlobalBank\Services directory.

Deployment View

The default installation of the Global Bank Internet banking application is configured for a development environment where the Presentation tier, services, and database are deployed on a single server. The Web services are configured to use inproc dispatching to communicate between the Service Interface and Service Implementation.

An alternative deployment configuration could achieve greater security by separating the Service Interface from the Service Implementation by using multiple servers separated by a firewall. This is often used where Web services are accessed by clients over a public network. Figure 5 shows an external Web server accessing a service interface deployed within a perimeter network.

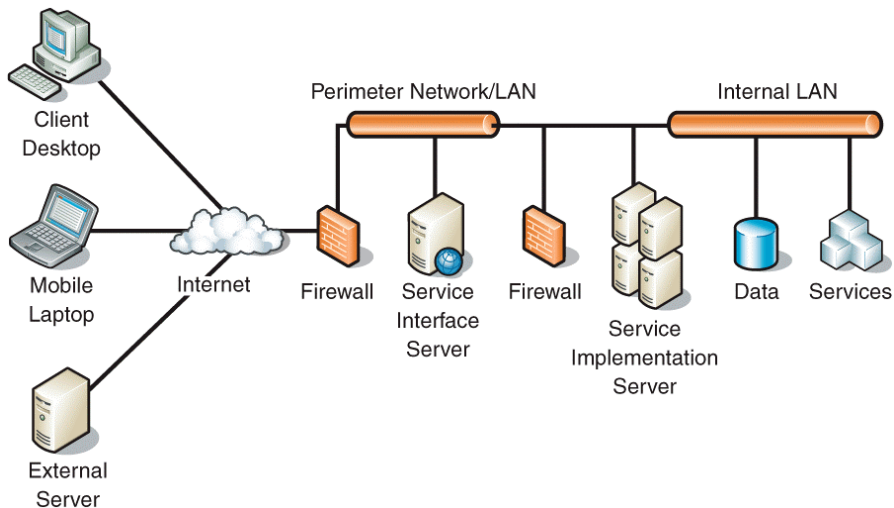


Figure 5
EDAF deployment view

Users access the Global Bank Internet banking application from desktops or laptops. From their interaction with the Web application, requests are generated and sent through the Web service transport to the Service Interface server. This server receives the request, invoking its pipeline. The request continues through the second firewall to the Service Implementation server, which ultimately invokes the business action. After this completes, the results are returned to the Web server, and then displayed to the user.

All of the assemblies for the Web application are deployed to the server for the Global Bank Internet banking application. The following assemblies are deployed:

- WebApplicationUI.dll
- WebApplication.UIProcess.dll
- WebApplication.MessageAdapters.dll
- WebApplication.Utilities.dll
- WebApplication.Common.dll
- WebApplication.WebServiceProxies.dll

Architecture Properties View

Architecture properties were considered throughout the development of the EDRI. Below are some considerations on security and localization.

Security

Security is comprised of several facets, but the two discussed here are authentication and authorization. Authentication is the process of uniquely identifying a user by verifying his or her credentials when the user attempts to make a connection to your application or service.

After users are authenticated, you can determine what they have access to within the system by using authorization. Authorization is confirmation that an authenticated user has permission to perform an operation. Authorization governs the resources (for example, files and databases) that an authenticated user can access and the operations (for example, changing passwords or deleting files) that an authenticated user can perform. Some users may be granted all of the privileges. This means they can access the entire functionality of the system. Others may be granted fewer privileges, meaning there may be some functions they are not permitted to use.

Security is an essential practice for building a Web service. For complete guidance, see the following resources:

- *Building Secure Microsoft ASP.NET Applications: Authentication, Authorization, and Secure Communication*, Redmond: Microsoft Press, 2003
- *Improving Web Application Security: Threats and Countermeasures*, Redmond: Microsoft Press, 2003

A later release of Global Bank will use the Web Services Enhancements toolkit to add additional security capabilities such as message level encryption and data integrity. This release assumes that the system would rely largely on a trusted subsystem model using Windows integrated security, SSL, and IPSec.

Authentication

You should use SSL to secure the data transported between the client browser and the Presentation tier. This is not implemented on the default installation. For more information, see [How To: Use SSL to Secure Communication with SQL Server 2000](#) in *Building Secure Microsoft ASP.NET Applications: Authentication, Authorization, and Secure Communication*.

After a user's credentials are validated, the user's identity (login ID) is stored in session state. The identity is then supplied to the back-end services for data entitlement. In other words, authentication between the Presentation tier and the application server will rely on a trusted subsystem model where the application server validates the calling server's credentials — not those of the end user initiating the request. This is not implemented on the default installation. For more information, see "The Trusted Subsystem Model" in [Chapter 3: Authentication and Authorization](#) in *Building Secure Microsoft ASP.NET Applications: Authentication, Authorization, and Secure Communication*.

The user's identity will be retrieved from the session state on the Presentation tier. It is passed to the receiving service to allow data entitlement rules to be executed, thus ensuring that users will be able to access only their own resources.

Authorization

Currently, all users are allowed to perform functionality provided by the Global Bank Internet banking application. In this version of the EDRI, no authorization has been implemented.

Localization

Global Bank is hoping to expand to non-English speaking countries in the near future. In this version of the EDRI several techniques were used to help make the transition easier. The following techniques were used for the Web application:

- Text displayed within the UI elements and error messages are stored in resource files.
- Cascading style sheets (CSS) are used in parts of the UI to enable support for localization.

For services, error messages and messages returned to the consumer of the services are stored in resource files.

For more details, see [Chapter 7: Globalization and Localization](#) in "Design and Implementation Guidelines for Web Clients."

3

Services

The services in the EDRI are designed to simulate interfaces into legacy systems. In many cases these services would call existing components that might, for example, have been developed in Microsoft Visual Basic® 6.0 development system to interact with the Microsoft Host Integration Server which would then provide access to legacy mainframe transactions.

The implementation of each service uses several databases to manage the data. The business action classes are designed to interact with the core database using the Microsoft Data Access Application Block. The end result is a lightweight implementation that allows us to simulate the interaction with legacy systems.

As previously described, the Global Bank Internet banking application uses a set of services to perform operations. This section focuses on the high-level design and implementation details of each service deployed in the Global Bank enterprise environment. However, before describing the services, we need to review common infrastructure support, common design views, and documentation standards.

Prior to developing services, an infrastructure was created that includes common XML schema types and custom handlers. In addition, there are elements of the EDAF that are common to all of the services. These elements include a default Service Implementation pipeline and use of SOAP message headers. For an overview of the EDAF, see “Appendix A— Inside the Enterprise Development Application Framework.”

Typically each service would be designed independently and there would be separate design documents for each one. Because this is a reference implementation with simulated services that would be impractical. Instead, we will discuss common design patterns before describing each service. The description of each service will contain some design views along with client interface information. The documentation structure and definition of messages will also be discussed prior to describing each service.

Service Infrastructure

Because Global Bank is using the EDAF to standardize the development of services, there are some common capabilities that can be applied to services. These capabilities are described in this section.

Request Message Validation

All of the services used by the Global Bank Internet banking application also use the **SyntaticValidation** handler to validate request messages. As a result, each request has an associated XML schema file used for the validation.

These files are located in the <Installation Location>\Microsoft EDRA\CS\ReferenceImplementation\GlobalBank\Services\Schemas folder. For detailed information on the **SyntaticValidation** handler, see Chapter 11, “Handlers Reference” in the EDRA documentation.

Any type restrictions defined in these schema files will also be described in the “Messages” section in each service description. The types described in Table 1 apply to the messages used by the different services to describe and validate input data.

Table 1: XML Schema Types

Name	Min/Max Length	Description
AccountNumberType	1/32	Uses a regular expression to restrict the first character to letters and decimal digits. All other characters are restricted to letters, decimal digits, dashes, and underlines.
DescriptionType	0/200	Uses a regular expression to restrict all characters to letters, decimal digits, spaces, and the following characters: _ \$ * % @ ! + = / \ () ? [] & " .
EntityNameType	0/50	Uses a regular expression to restrict the first character to letters and decimal digits. All other characters are restricted to letters, decimal digits, dashes, underlines, spaces, and ampersands (&).
ExternalIdentifierType	1/32	Uses a regular expression to restrict the first character to letters and decimal digits. All other characters are restricted to letters, decimal digits, dashes, and underlines.
PasswordType	1/16	Uses a regular expression to restrict all characters to letters, decimal digits, and underlines.
UserNameType	1/32	Uses a regular expression to restrict the first character to letters and decimal digits. All other characters are restricted to letters, decimal digits, dashes, and underlines.

These types are used to define **String** types that have additional restrictions. For instance, instead of defining an element named `FirstName` as a **String** type in the message, we would use **UserNameType** instead. This would define the element as a string with the restrictions defined above.

Note: It is possible to optimize performance of services by replacing XSD validations with C# or Visual Basic .NET code that performs similar validations.

Custom LoggingHandler

This is an atomic handler used to write log entries to the Microsoft SQL Server™ database named **GlobalBank_Core**. Handlers are used to implement cross cutting concerns. The custom **LoggingHandler** uses application configuration information to get a connection string and then writes request message information to the database.

When To Use

Because this is an atomic handler, it could be used anywhere in the pipelines; however, it will only log request messages and handlers that are specific to the application should generally be used in the Service Implementation pipeline. For more information about exceptions in handlers, see Chapter 11, “Handlers Reference” in the EDRA documentation.

Configuration

To use this handler you will need to add an application configuration entry for the database connection. Currently this information has been added to all of the Web service transports used by the Global Bank services, as shown in the following code.

```
<appSettings>
  <add key="AppDBConnectionString"
        value="Initial Catalog=GlobalBank_Core;Data Source=localhost;
              User Id=ReferenceImplementationSystemAccount;
              Password=ServicesRI$31687#" />
</appSettings>
```

If this entry does not exist and the **Logging** handler is used, the handler will throw an exception.

Note: You can use Windows integrated security to modify the EDRI configuration files to access SQL Server. However, you must set up the <Machine Name>\ASPNET account in Windows XP. For Windows 2003, use the Windows NT Authority\Network Service account as a user on SQL Server and grant the appropriate permissions. For information, see [Accessing SQL Server Using Windows Integrated Security](#) on MSDN.

When using this handler in a pipeline, all you need to do is add it to the before group of the pipeline as shown in the following code.

```
<before>
  <handler handlerName="LoggingHandler"/>
</before>
```

Implementation

When this handler is added to a pipeline, request messages will be serialized into a string using Base64 encoding and written to the **CustomerActivitySecurityLog** table of the **GlobalBank_Core** database.

Figure 1 shows an example entry.

Id	Channel	CustomerLoginId	LogDate	SerializedContextInformation
17	Home Banking	11111111	2004-05-21 10:54:11.810	SubscribeToPayBill - PFNPQVatRUSWokVudm...
24	Home Banking	11111111	2004-05-21 10:54:21.913	PerformFundsTransfer - PFNPQVatRUSWokVu...

Figure 1

Example log entry

Notice that the **SerializedContextInformation** contains the name of the service action followed by a string of characters. These characters represent the Base64 encoded request message. To read these entries, you will need to decode the message data.

Service Interface and Service Implementation Pipelines

The Service Interface and Service Implementation pipelines are configured in the **GlobalBankServices.config** file. The Service Interface pipeline for all of the services in the EDRI uses the following default configuration.

```
<pipeline name="Default"
  transportName="*"
  serviceActionName="*"
  targetName="*" />
```

Typically, the **transportName** is one of the following: **InProcTransport**, **WebServiceTransport**, **MessageQueueTransport**, or **RemotingTransport**. In the EDRI, however, all the services use the Web Service Interface Transport because the EDRI only supports the Web Service Interface Transport out of the box. The **serviceActionName** attribute provides the name of a specific service action that the pipeline is associated with. The **targetName** for the Service Interface pipeline is typically any of the dispatching transports (or possibly the business action target).

The Service Implementation pipeline for all of the services in the EDRI uses the following default configuration.

```
<pipeline name="Default"
    serviceActionName="*"
    targetName="businessAction"/>
```

Handlers can be defined within the **<before>** and **<after>** sections of the Service Interface and Service Implementation pipelines. Generally, you should use handlers that are transport dependent in the service interface pipeline and handlers that are specific to your application in the service implementation pipeline. The **targetName** for the Service Implementation pipeline is typically the business action target.

Web Service Headers

All of the services in the EDRI are implemented using Web service transports. Any header information passed into these services needs to be added as a SOAP header. Fortunately, the EDAF defines a **RequestHeader** class that extends the **SoapHeader** class provided by the Microsoft .NET Framework. This class provides a public field named **Headers** that is defined as an array of objects. To add header information, you must first initialize this collection. Each header is then added as another array of objects that contains the key and value.

The following code example shows how to add a single header with **UserName** and **Channel** information.

```
RequestHeader requestHeaders = new RequestHeader();
requestHeaders.Headers = new object[2];
object[] userName = new object[2];
userName[0] = "UserName";
userName[1] = "11111111";
object[] channel = new object[2];
channel[0] = "Channel";
channel[1] = "Home Banking";
requestHeaders.Headers[0] = userName;
requestHeaders.Headers[1] = channel;
```

In this example we initialized the **Headers** collection as an array containing two objects. Next we initialized two additional arrays, added key/value information, and then added the new arrays to the **Headers** collection.

Common Design Views

There are two views that are common to all services in the Global Bank system. One is a deployment view that describes how the services are deployed within the enterprise. Another is the policy view, which describes authentication and transport level security used by the services.

Deployment View

The diagram in Figure 2 shows the main systems used to support Global Bank services. In addition, this diagram describes the network topology, which includes subnet boundaries and firewalls.

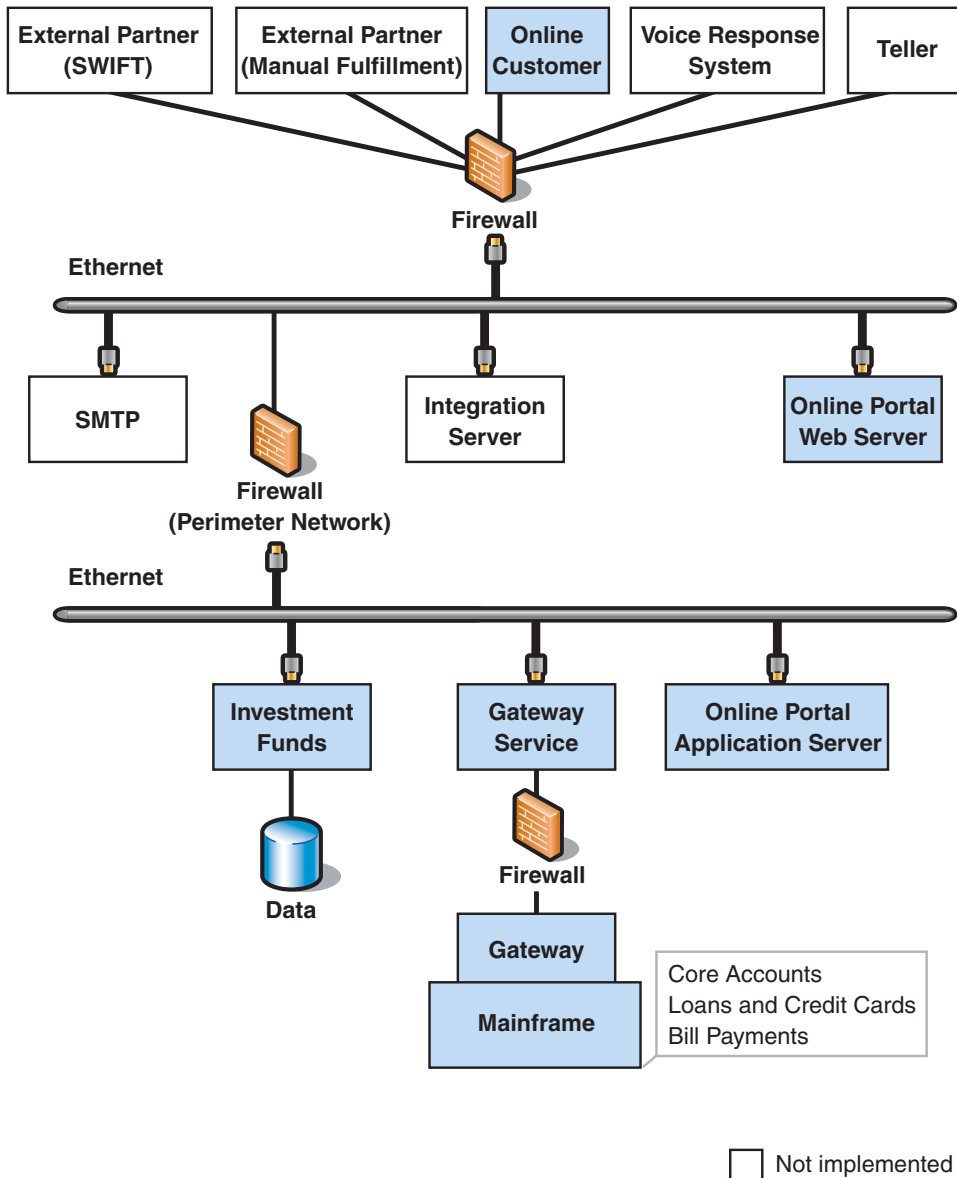


Figure 2
Enterprise network diagram

Because the services use Web services, the assemblies for each one will be located in the Bin folder under the Project folder. In addition, clients should not access these assemblies directly; instead, the clients should use Web service proxies to access the service. As a result, it is not necessary to list the service assemblies.

Policy View

The policy view provides information about the security requirements for this service. The information provided includes authentication and transport security.

Service Authentication

The services proposed in this chapter are for internal use only. Network level security such as firewalls and private IP addresses prevent external access to these services.

In the first release of this solution, the Presentation tier will be the only client using the Web services. Authentication between the Presentation tier and the application server will rely on a trusted subsystem model where the application server validates the calling server's credentials — not those of the end user initiating the request. For more information, see “The Trusted Subsystem Model” in [Chapter 3: Authentication and Authorization](#) in *Building Secure Microsoft ASP.NET Applications: Authentication, Authorization, and Secure Communication*.

The end user's identity will be retrieved from session state on the Presentation tier and passed to the receiving service to allow data entitlement rules to be executed — thus ensuring that end users are able to access only their own data.

As mentioned in the “Security” topic in the “Non-Functional Application Considerations” section, an upcoming update to the voice response system will use the same set of services. In addition, a scenario for teller terminals based on smart client technologies will also use the same set of services. At that point, a token-issuing service will be developed and a session ID will be passed to the service instead of the user's identity.

The Global Bank Internet banking application will use the Enterprise Development Application Framework (EDAF), which simplifies this process because the **Identity** handler will be replaced with a custom token authentication handler and no changes to the services interface or business logic will be required.

Transport Layer Security

Interactions between the client and the Presentation tier are assumed to occur using SSL/TLS with server certificates. Security between the Presentation tier and the application servers hosting the business logic would typically include network level security using SSL and IPsec. SSL is used to encrypt the transport. IPsec is used to restrict access so that only Presentation tier servers access the application servers. The application servers will use Windows integrated security to authenticate the calling servers. SSL client certificates are also being considered as an additional means of allowing the application server to validate the credentials of the Presentation tier.

Service Documentation

Because the Service Implementations are simulated, we will not attempt to create standard design documents. Instead, we will provide a high-level description of the simulated design, along with information used to interact with the service from a client.

The Logical view within each service includes an activity diagram that shows operations performed by the service, along with information about the Web Service Interface. The “Client Interface” section focuses on message definitions, along with configuration of the service and a description of handlers used.

Messages

The example in Figure 3 shows a class diagram followed by several tables that describe a generic response message.

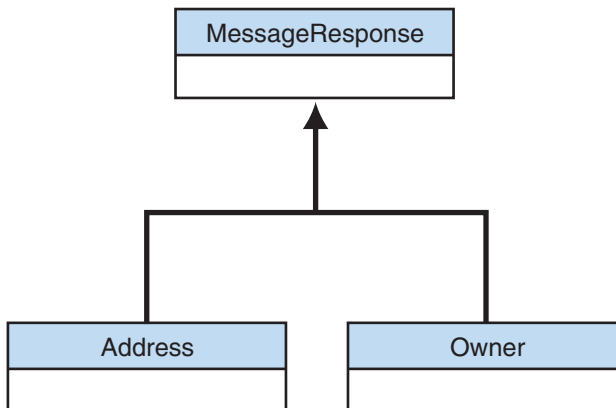


Figure 3

Sample response message

The following tables provide information about the XSD data types and element names used to define the response message.

Table 2: MessageResponse Type

Element	Type	Description
Addresses	AddressCollection	Collection of addresses
OwnerInfo	Owner	Owner information

Table 3: Address Type

Element	Type	Description
LineOne	String	Line one of the address
LineTwo	String	Line two of the address
State	String	State
Country	String	Country
Zip	String	Zip code

Table 4: Owner Type

Element	Type	Description
FirstName	String	Owner's first name
LastName	String	Owner's last name
MiddleInitial	String	Owner's middle initial

The information in the preceding diagram and tables can be mapped to an XML structure by using the type and element names. The following is an example of what the XML would look like for our **MessageResponse** type.

```

<MessageResponse>
  <Addresses>
    <Address>
      <LineOne>123 Main Street</LineOne>
      <LineTwo></LineTwo>
      <State>MD</State>
      <Country>USA</Country>
      <Zip>12345</Zip>
    </Address>
    <Address>
      <LineOne>228 Pleasant Street</LineOne>
      <LineTwo></LineTwo>
      <State>VA</State>
      <Country>USA</Country>
      <Zip>67890</Zip>
    </Address>
  </Addresses>
  <OwnerInfo>
    <FirstName>Gytis</FirstName>
    <LastName>Barzdukas</LastName>
    <MiddleInitial>M</MiddleInitial>
  </OwnerInfo>
</MessageResponse>

```

Notice that the **Addresses** element of the **MessageResponse** type contains multiple **Address** elements. When looking at the type definition for **Addresses** in Table 3, you will see that it is defined as an **AddressCollection**. Any time you see collection types, the element name is used to define the collection element and the type name is used for each child element in the collection.

The **OwnerInfo** element represents a single **Owner** type, which is a child element of the **MessageResponse**. Notice that the **OwnerInfo** type in Table 2 is a reference to the **Owner** type described in Table 4. As with the collection, the element name is used to define the XML element. The difference here is that the type name is not used.

Conceptual View

Global Bank's chief architect stated two strategic goals for the Global Bank Internet banking application architecture:

- Standardize the development of services using an enterprise framework to ensure consistent and timely development of services.
- Consider WS-I conformant Web services as a means to expose system functionality so that other channels, such as Teller Terminals, Voice Response Systems and ATMs, can use the same functionality at a later point.

The Global Bank Internet banking application uses the Microsoft Service-Oriented Integration pattern and the EDAF to help meet these requirements. In addition to using WS-I conformant Web services, the Service-Oriented Integration pattern recommends using the Service Interface pattern to separate transport specific service interface mechanics from application logic. This pattern is fundamental to the design of the EDAF. Figure 4 illustrates service design using the Service-Oriented Integration pattern. Appendix A provides the conceptual view of the EDAF.

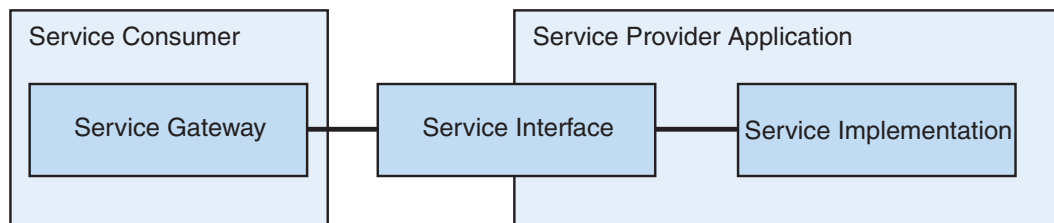


Figure 4

Separation of Service Interface and Service Implementation

See the following patterns for more information:

- [Service-Oriented Integration](#)
- [Service Interface](#)

AccountStatement Service

This service provides a single action that returns all of the user's account statement information. When the service action is invoked the service interacts with legacy systems, through several different interfaces, which includes another service.

Logical View

The EDAF requires each service to define a Service Interface transport and a business action component. For details about the framework, see the [Enterprise Development Reference Architecture](#) documentation. In addition, the "Funds Transfer Walkthrough" section describes how the framework processes a service request.

The Service Interface transport used for this service is a Web service named **AccountStatementService**, as follows:

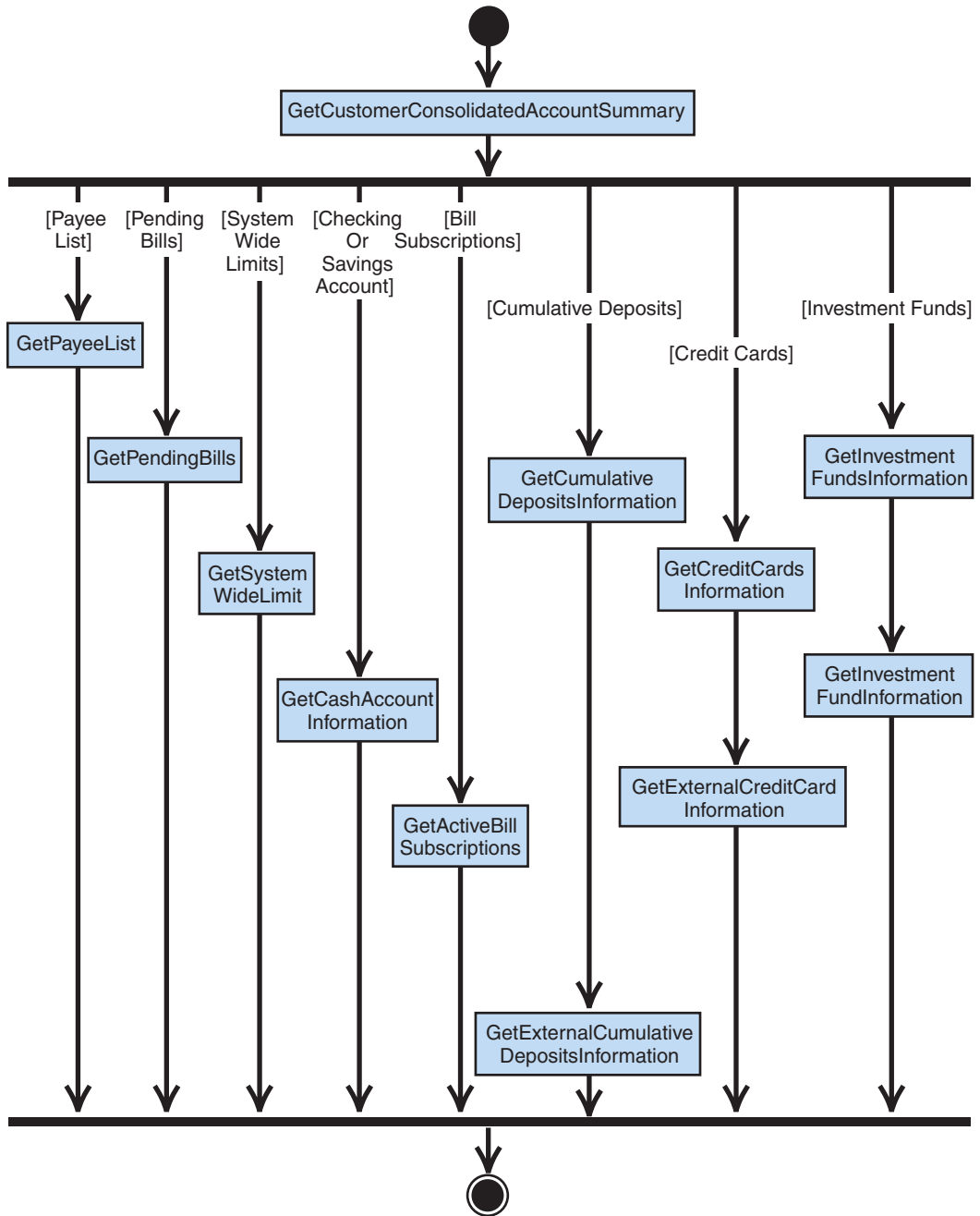
AccountStatementService
+Credentials : RequestHeader
+GetCustomerConsolidatedAccountStatement(VoidInput : VoidInput) : ConsolidatedAccountStatementResponse

The **AccountStatementService** class defines a public method named **GetCustomerConsolidatedAccountStatement**, which takes a **VoidInput** as the request parameter and returns a **ConsolidatedAccountStatementResponse**. This class also defines a public attribute named **Credentials**, which is a SOAP request header.

The SOAP header should contain the following values:

Name	Value
UserName	Name of the user.
Channel	Identifies the application used to make the request.

The business action class for the **AccountStatement** service is named **AccountStatementBusinessAction**, which also has a method named **GetCustomerConsolidatedAccountStatement**. The business action method is responsible for collecting account information as shown in the activity diagram in Figure 5. The account information is obtained asynchronously.

**Figure 5**

GetCustomerConsolidatedAccountStatement activity diagram

As you can see, there are many parallel actions that are performed by this business action. In most cases the service will not attempt to gather data if the user does not have a particular type of account or data for an account.

Client Interface

To interact with the service, a client will need to know the structure of messages, the configuration information of this service, and the handlers that it will use.

Messages

This service uses a common type named **VoidInput** as the request parameter and returns a type named **ConsolidateAccountStatementResponse**, which contains several other types.

Request Message — **VoidInput**

This is defined as a common message type used by any service that does not have input parameters. This type does not contain any values.

ResponseMessage — **ConsolidatedAccountStatementResponse**

This type contains many other types, which contains account statement information.

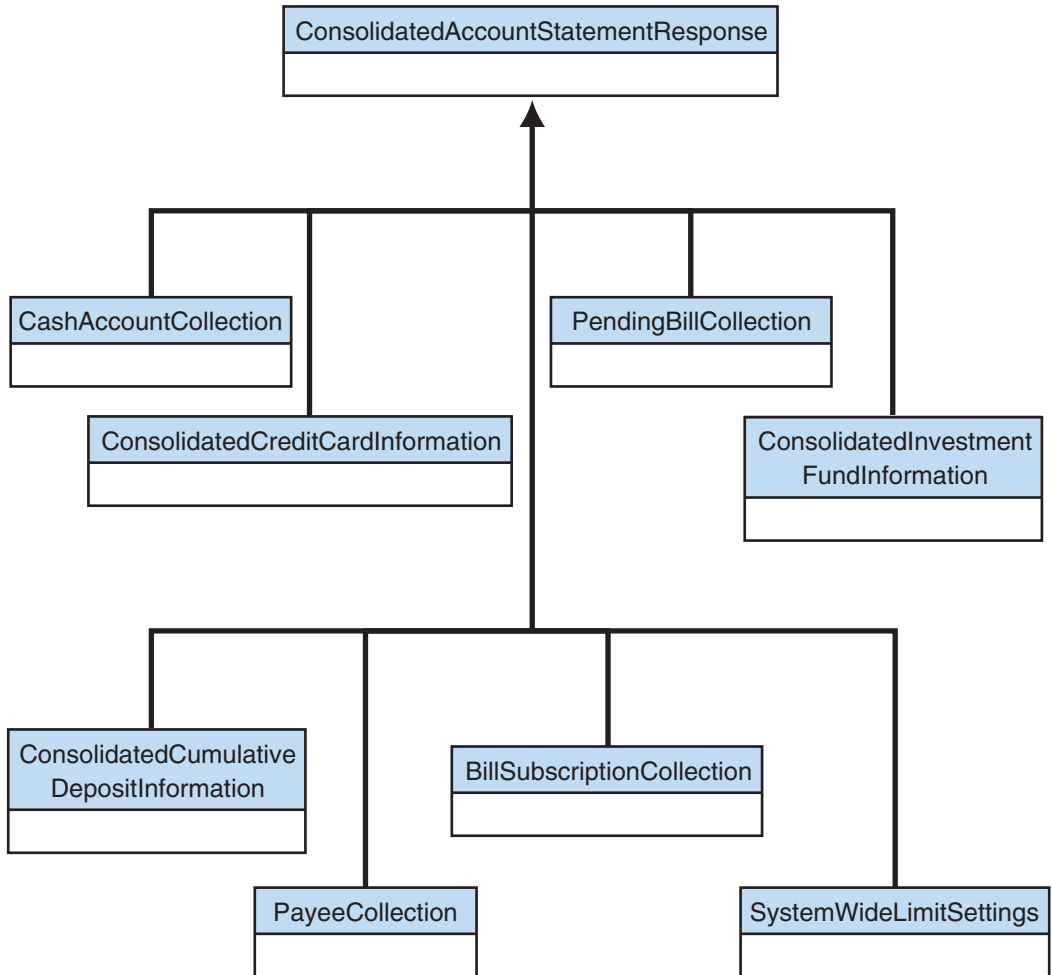


Figure 6
ConsolidatedAccountStatementResponse

The following tables provide information about the types and element names used to define the response message.

Table 5: ConsolidatedAccountStatementResponse

Element	Type	Description
CashAccounts	CashAccountCollection	Cash accounts
PendingBills	PendingBillCollection	Pending bills
PayeeList	PayeeCollection	Payees
SubscribedBills	BillSubscriptionCollection	Bill subscriptions
SystemWideLimits	SystemWideLimitSettings	System settings
ConsolidatedCreditCardInfo	ConsolidatedCreditCard Information	Credit cards
Consolidated Cumulative DepositInfo	ConsolidatedCumulative DepositInformation	Cumulative deposits
Consolidated InvestmentFundInfo	ConsolidatedInvestment FundInformation	Investment funds

Table 6: CashAccount

Element	Type	Description
AccountNumber	string	Number assigned to this account
ProductType	Int	Type of product
Balance	decimal	Balance for this account
AccountType	string	Type of account

Table 7: CreditCard

Element	Type	Description
CreditCardNumber	string	Credit card number
Type	string	Type of credit card
Balance	decimal	Balance on credit card
PaymentDueDate	dateTime	Date payment is due
ExpirationDate	dateTime	Credit card expiration date
NameOnCard	string	Name on the credit card
CreditLimit	decimal	Maximum credit allowed

Table 8: PendingBill

Element	Type	Description
BillIdInBillingSystem	string	ID used by billing system
Amount	decimal	Amount of the bill
DueDate	dateTime	Date the bill is due
PayeeIdentifier	Int	Link to payee information
CustomerIdInBillingSystem	string	Customer ID used by billing system
PayeeName	string	Name of the payee

Table 9: InvestmentFund

Element	Type	Description
InvestmentFundIdentifier	string	ID of this investment fund
Name	string	Name of the fund
Shares	decimal	Number of shares
Quote	decimal	Quoted price of the fund

Table 10: CumulativeDeposit

Element	Type	Description
CumulativeDepositIdentifier	string	ID for this record
PrincipalAmount	decimal	Principal amount of deposit
InterestAmount	decimal	Interest amount of deposit
TermDurationInDays	Int	Duration in days
ActionOnDueDate	string	Action to take when due
DueDate	dateTime	Date the deposit is due

Table 11: BillSubscription

Element	Type	Description
CustomerExternalIdentifier	string	ID of the customer
AccountActive	boolean	Flag indicating active state
PayeeId	Int	Link to payee information
PayeeName	string	Name of the payee

Table 12: SystemWideLimitSettings

Element	Type	Description
Channel	string	Identifies the application such as “Home Banking”
SingleBillPaymentLimit	decimal	Maximum amount for a bill
SingleFundTransferLimit	Decimal	Maximum amount that can be transferred
MonthlyFundTransferLimit	Decimal	Maximum amount that can be transferred in a month

Service Configuration

The Service Interface pipeline configuration in GlobalBankServices.config is shown in the following code.

```
<pipeline name="AccountStatementPipeline"
  transportName="WebServiceTransport"
  serviceActionName="GetCustomerConsolidatedAccountStatement"
  targetName="inproc">
  <before>
    <handler handlerName="ExecutionTimeout">
      <timeoutConfiguration timeout="300"/>
    </handler>
    <handler handlerName="SyntacticValidation">
      <syntactValidationSettings
        requestSchema="C:\Program Files\Microsoft EDRA\CS\
          ReferenceImplementation\GlobalBank\Services\Schemas\
          ConsolidatedAccountStatementRequest.xsd"
        responseSchema=""/>
      </handler>
    <handler handlerName="Identity"/>
  </before>
  <after>
    <handler handlerName="SyntacticValidation">
      <syntactValidationSettings
        requestSchema="C:\Program Files\Microsoft EDRA\CS\
          ReferenceImplementation\GlobalBank\Services\Schemas\
          ConsolidatedAccountStatementRequest.xsd"
        responseSchema=""/>
      </handler>
    <handler handlerName="ExecutionTimeout"/>
  </after>
</pipeline>
```

Note: The **requestSchema** path information in the preceding code is displayed on multiple lines due to formatting restrictions. If this configuration is copied, you will need to modify the schema path information so that it is all on one line.

Notice that the **serviceActionName** matches the name of the public method shown in the **AccountStatementService** class. The EDAF uses the method name to look for pipeline and business action entries with the same **serviceActionName**. This pipeline is also configured to use a Web service transport, an **inproc** target, an **ExecutionTimeout** handler, and a **SyntacticValidation** handler.

The Service Implementation pipeline configuration in `GlobalBankServices.config` is shown in the following code.

```
<pipeline name="AccountStatementImplementationPipeline"
  serviceActionName="GetCustomerConsolidatedAccountStatement"
  targetName="businessAction">
  <before>
    <handler handlerName="LoggingHandler"/>
    <handler handlerName="AppInstrumentation"/>
  </before>
  <after>
  </after>
</pipeline>
```

The Service Implementation pipeline uses the **LoggingHandler** and the **AppInstrumentation** handlers, and executes the business action. When the **businessAction** target is invoked, the framework looks in the `GlobalBankServices.config` file for a **businessAction** element with the same **serviceActionName** attribute, which is shown in the following code.

```
<businessAction
  serviceActionName="GetCustomerConsolidatedAccountStatement"
  type="Microsoft.ReferenceImplementation.Services.AccountStatement.
    BusinessActions.AccountStatementBusinessAction,
    AccountStatement.BusinessActions"
  invocationMethod="Serialization">
  <request type="Microsoft.ReferenceImplementation.Services.
    CommonMessageDefinitions.VoidInput,
    Services.CommonMessageDefinitions">
    <method name="GetCustomerConsolidatedAccountStatement"/>
  </request>
  <response
    type="Microsoft.ReferenceImplementation.Services.AccountStatement.
      MessageDefinitions.ConsolidatedAccountStatementResponse,
      AccountStatement.MessageDefinitions"/>
</businessAction>
```

Note: The type information shown in the preceding code was broken across multiple lines due to formatting constraints. If this configuration is copied, you will need to modify the type information so that it is all on one line.

Handlers

The following handlers are used by this service:

Table 13: AccountStatement Handlers

Name	Description
ExecutionTimeout	Aborts the current thread if execution time exceeds 300 seconds.
SyntacticValidation	Uses the ConsolidatedAccountStatementRequest.xsd schema to validate the request message. Not configured to validate a response message.
Identity	Extracts the UserName from the message header and creates an authenticated GenericPrincipal using the name.
LoggingHandler	Writes log entries to the GlobalBank_Core database.
ApplInstrumentation	Updates the following Performance Monitor counters: ReferenceArchitecture.TotalRequests ReferenceArchitecture.RequestsPerSecond

Deployment View

The **AccountStatement** service is located within the Global Bank system. The service uses internal resources as well as information from external systems when gathering statement information. The deployment diagram in Figure 7 shows the different systems involved.

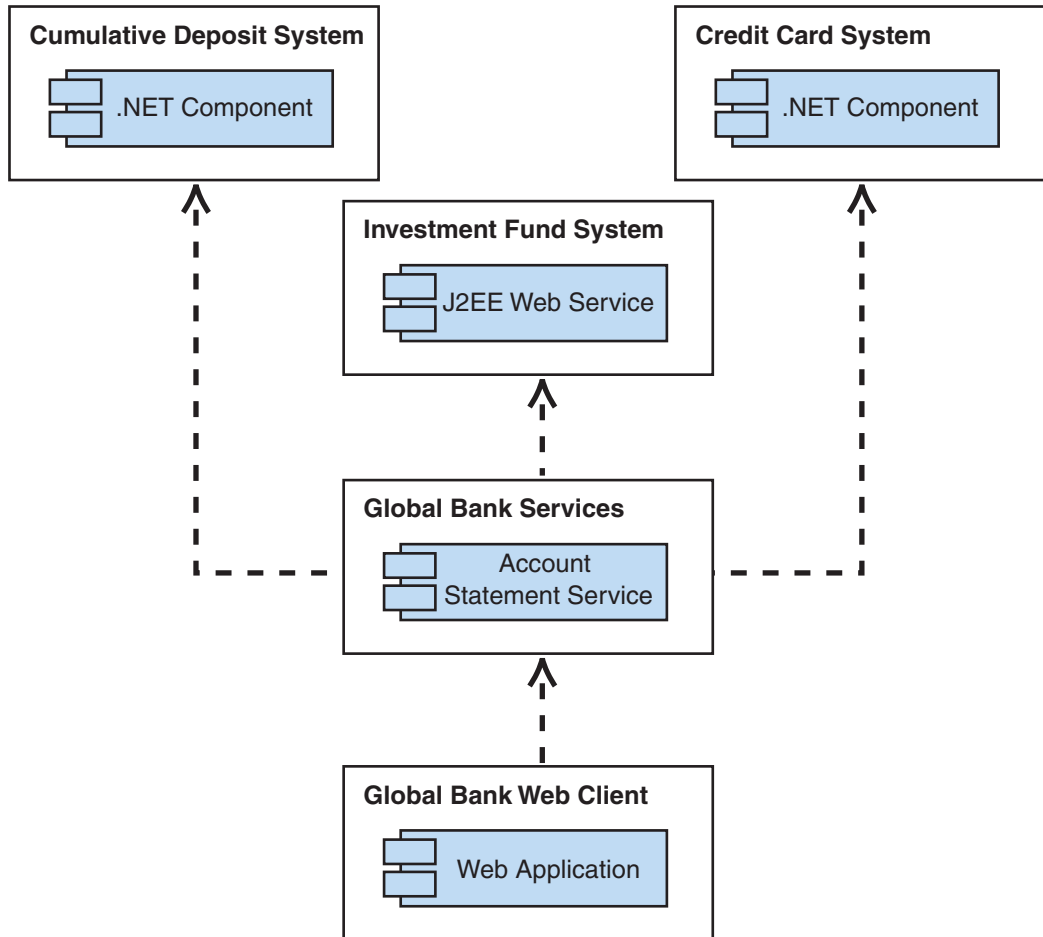


Figure 7

AccountStatement service deployment diagram

Note: The Investment Fund System has been developed as a standard .asmx Web service with the intention of simulating a service running on another platform; therefore, handlers have not been applied to it. Security would rely on a trusted subsystem model using either SSL or IPsec.

Authentication Service

This service provides a single point for authenticating users of the Global Bank Internet banking application.

Logical View

The Enterprise Development Application Framework requires each service to define a Service Interface transport and a business action component. For details about the framework, see the [Enterprise Development Reference Architecture](#) documentation. In addition, the “Funds Transfer Walkthrough” section describes how the framework processes a service request.

The Service Interface transport used for this service is a Web service named **AuthenticationService**, as follows:

AuthenticationService
+Authenticate(AuthenticationRequest : AuthenticationRequest): AuthenticationResponse
+DisableAccount(AuthenticationRequest : AuthenticationRequest): VoidResponse

The **AuthenticationService** class defines two public methods named **Authenticate** and **DisableAccount**, both of which take an **AuthenticationRequest** as the request parameter. The **Authenticate** method returns an **AuthenticationResponse** while the **DisableAccount** method returns a **VoidResponse**. This service does not use any header information.

The business action class for the **AuthenticationService** is named **LoginBusinessAction**. This class also defines two methods named **Authenticate** and **DisableAccount**, both of which take the same parameters as the authentication service methods. The business action class is responsible for authenticating the user as shown in the activity diagram in Figure 8.

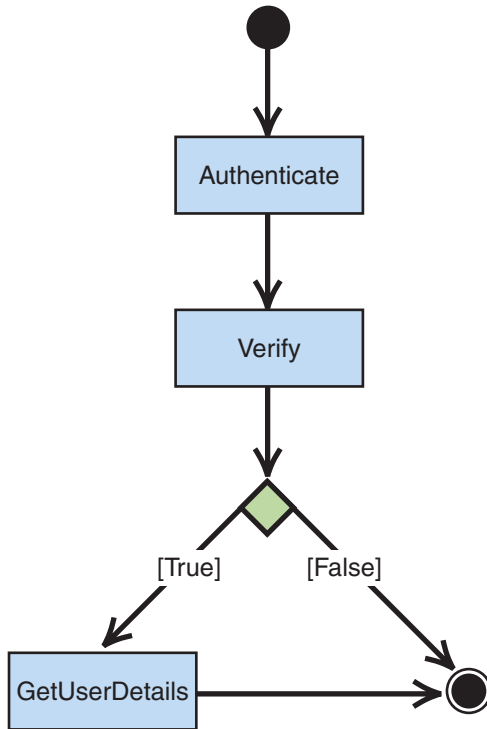


Figure 8
Authenticate activity diagram

When looking at the diagram, you can see that the first step is to verify the user. If the user is verified, the next step is to retrieve user detail information.

The **DisableAccount** action does not perform any additional steps other than disabling an account and returning.

Client Interface

To interact with the service, a client will need to know the structure of messages, the configuration information of this service, and the handlers that it will use.

Messages

This service uses an **AuthenticateRequest** message for the input parameters and returns the result in an **AuthenticationResponse** message.

Request Message — AuthenticationRequest

The **AuthenticationRequest** message uses a single type that contains the elements listed in Table 14.

Table 14: AuthenticationRequest

Element	Type	Description
UserName	UserNameType	User name used for authentication. This is the ATM card number (16 digits).
Password	PasswordType	Password used for authentication. This is the PIN number of 4 digits length. This restriction is placed on the client, not on the service. The service allows up to 16 characters.
Channel	EntityNameType	Channel used to access the banking information. This identifies the application such as “Home Banking” used to make the request.
AllowedPasswordRetryLimit	Int	The maximum number of password retries allowed.

Response Message — AuthenticationResponse

The **AuthenticationResponse** message is also a single type that contains the elements listed in Table 15.

Table 15: AuthenticationResponse

Element	Type	Description
ReturnValue	Boolean	Value that determines if the authentication was successful or not. Allowed value is true/false.
UserFirstName	String	User's first name.
UserLastName	String	User's last name.
LastLoginDate	DateTime	Last login date for the user.

Response Message — VoidResponse

This is defined as a common message type use by any service that does not return a response. This type does not contain any values.

Service Configuration

The Service Interface pipeline configuration in `GlobalBankServices.config` is shown in the following code.

```
<pipeline name="AuthenticationPipeline"
  transportName="WebServiceTransport"
  serviceActionName="Authenticate"
  targetName="inproc">
  <before>
    <handler handlerName="ExecutionTimeout">
      <timeoutConfiguration timeout="300"/>
    </handler>
    <handler handlerName="SyntacticValidation">
      <syntactValidationSettings
        requestSchema="C:\Program Files\Microsoft EDRA\CS\
          ReferenceImplementation\GlobalBank\Services\Schemas\
          AuthenticationRequest.xsd"
        responseSchema=""/>
      </handler>
    </before>
    <after>
      <handler handlerName="SyntacticValidation">
        <syntactValidationSettings
          requestSchema="C:\Program Files\Microsoft EDRA\CS\
            ReferenceImplementation\GlobalBank\Services\Schemas\
            AuthenticationRequest.xsd"
          responseSchema=""/>
        </handler>
      <handler handlerName="ExecutionTimeout"/>
    </after>
  </pipeline>
```

Note: The **requestSchema** path information in the preceding code is displayed on multiple lines due to formatting restrictions. If this configuration is copied, you will need to modify the schema path information so that it is all on one line.

Notice that the **serviceActionName** matches the name of the public method shown in the **AuthenticationService** class. The EDAF uses the method name to look for pipeline and business action entries with the same **serviceActionName**. This pipeline is also configured to use a Web service transport, an **inproc** target, an **ExecutionTimeout** handler, and a **SyntacticValidation** handler.

The Service Implementation pipeline configuration in GlobalBankServices.config is shown in the following code.

```
<pipeline name="AuthenticationImplementationPipeline"
  serviceActionName="Authenticate" targetName="businessAction">
  <before>
    <handler handlerName="AppInstrumentation"/>
  </before>
  <after>
  </after>
</pipeline>
```

The Service Implementation pipeline uses the **AppInstrumentation** handler and executes the business action. When the **businessAction** target is invoked, the framework looks in the GlobalBankServices.config file for a **businessAction** element with the same **serviceActionName** attribute, which is shown in the following code.

```
<businessAction
  serviceActionName="Authenticate"
  type="Microsoft.ReferenceImplementation.Services.Authentication.
    BusinessActions.LoginBusinessAction,
    Authentication.BusinessActions"
  invocationMethod="Serialization">
  <request type="Microsoft.ReferenceImplementation.Services.Authentication.
    MessageDefinitions.AuthenticationRequest,
    Authentication.MessageDefinitions">
    <method name="Authenticate"/>
  </request>
  <response type="Microsoft.ReferenceImplementation.Services.Authentication.
    MessageDefinitions.AuthenticationResponse,
    Authentication.MessageDefinitions"/>
</businessAction>
```

Note: The type information was broken across multiple lines due to formatting constraints. If this configuration is copied, you will need to modify the type information so that it is all on one line.

Handlers

Table 16 lists the handlers this service uses.

Table 16: Authentication Handlers

Name	Description
ExecutionTimeout	Aborts the current thread if execution time exceeds 300 seconds.
SyntacticValidation	Uses the AuthenticationRequest.xsd schema to validate the request message. Not configured to validate a response message.
ApplInstrumentation	Updates the following Performance Monitor counters: ReferenceArchitecture.TotalRequests ReferenceArchitecture.RequestsPerSecond

Deployment View

The **Authentication** service is located within the Global Bank system. The deployment diagram in Figure 9 shows the different systems involved.

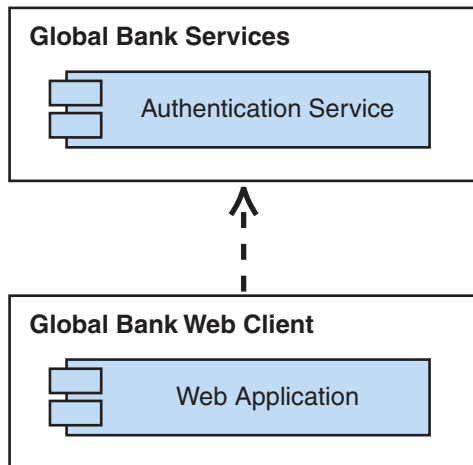


Figure 9

Authentication Service deployment diagram

BillPayment Service

The BillPayment service performs bill payment operations by initiating a funds transfer operation between the customer account and the bill payee collection account. This also includes a compensating transaction, which will undo the operation if an exception occurs.

Logical View

The Enterprise Development Application Framework requires each service to define a Service Interface transport and a business action component. For details about the framework, see the [Enterprise Development Reference Architecture](#) documentation. Appendix B, “Exploring the EDAF Using the Bill Payment Use Case” shows the high-level interactions between the client and EDAF components for the Bill Payment use case. In addition, the “Funds Transfer Walkthrough” section describes how the framework processes a service request.

The Service Interface transport used for this service is a Web service named **BillPaymentService**, as follows:

BillPaymentService
+Credentials : RequestHeader
+PayPendingBill(BillPaymentRequest : BillPaymentRequest) : BillPaymentResponse

The **BillPaymentService** class defines a public method named **PayPendingBill**, which takes a **BillPaymentRequest** as the request parameter and returns a **BillPaymentResponse**. This class also defines a public attribute named **Credentials**, which is a SOAP request header.

The SOAP header should contain the following values:

Name	Value
UserName	Name of the user.
Channel	Identifies the application used to make the request.

The business action class for the **BillPaymentService** is named **BillPaymentBusinessAction**. This class also defines a method named **PayPendingBill**, which takes the same parameters as the bill payment service and returns the same response. The business action class is responsible for validating account information, paying the bills, and rolling back transactions as shown in the activity diagram in Figure 10.

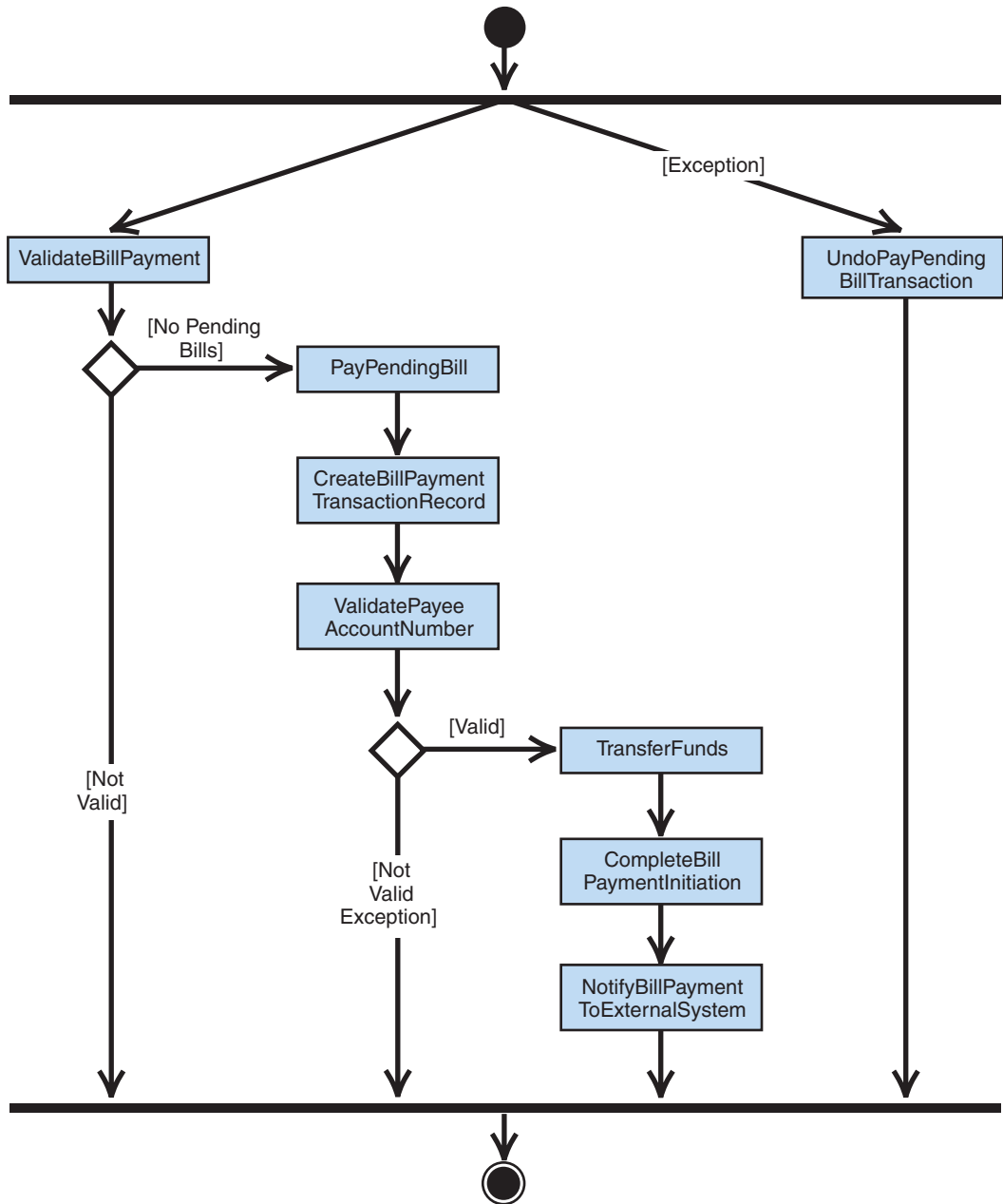


Figure 10
Bill Payment activity diagram

When looking at the activity diagram, you can see that there are three possible paths. The first action is to validate the bill payment information. If that is successful, the pending bill is paid; if it is not successful, a validation failure message is sent to the client. At any time while processing this request, an exception can be thrown. If that occurs, the **UndoPayPendingBillTransaction** action will be performed.

Client Interface

To interact with the service, a client will need to know the structure of messages, the configuration information of this service, and the handlers that it will use.

Messages

This service uses a **BillPaymentRequest** message for the input parameters and returns the result in a **BillPaymentResponse** message.

Request Message — **BillPaymentRequest**

The **BillPaymentRequest** message uses a single type that contains the elements listed in Table 17.

Table 17: BillPaymentRequest

Element	Type	Description
SourceAccountNumber	AccountNumberType	The account number identifying the source account for the bill payment.
Amount	Decimal	Bill amount to be paid.
CustomerExternalIdentifier	ExternalIdentifierType	User account number in the payee system.
PayeeIdentifier	Int32	Identifier for the payee corresponding to the bill.
PayeeFriendlyName	EntityNameType	Friendly name of the payee corresponding to the bill.
ExternalBillIdentifier	ExternalIdentifierType	Identifier for the bill in the external bill payment system.

Response Message — BillPaymentResponse

This type contains **CashAccount** information along with the response information, as shown in Figure 11.

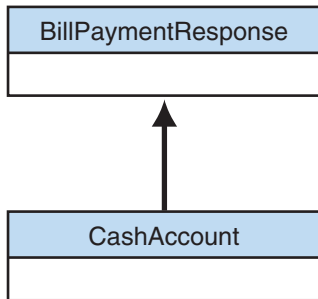


Figure 11

BillPaymentResponse

The following tables provide information about the types and element names used to define the response message.

Table 18: BillPaymentResponse

Element	Type	Description
SourceAccount	CashAccount	The source account for the bill payment.
BillPaymentinitiationSuccessful	Boolean	Indicates whether the bill payment initiation was successful.
FailureReason	String	If the bill payment initiation failed contains the reason for failure.
ReferenceCode	Int32	Reference code for the transaction that the user can use to track the transaction.

Table 19: CashAccount

Element	Type	Description
AccountNumber	String	Account number uniquely identifying the account.
ProductTypeeld	Int32	The integral value maps to indicates the type of the product, such as a savings account or checking account.
Balance	Decimal	The balance amount in the account.
AccountType	String	Type of account, such as checking account or savings account.

Service Configuration

The Service Interface pipeline configuration in GlobalBankServices.config is shown in the following code.

```
<pipeline name="BillPaymentPipeline"
  transportName="WebServiceTransport"
  serviceActionName="PayPendingBill"
  targetName="inproc">
  <before>
    <handler handlerName="ExecutionTimeout">
      <timeoutConfiguration timeout="300"/>
    </handler>
    <handler handlerName="SyntacticValidation">
      <syntactValidationSettings
        requestSchema="C:\Program Files\Microsoft EDRA\CS\
          ReferenceImplementation\GlobalBank\Services\Schemas\
          BillPaymentRequest.xsd"
        responseSchema=""/>
    </handler>
    <handler handlerName="Identity"/>
    <handler handlerName="DuplicateMessage"/>
      <duplicateHandlerSettings lifeTime="60"
        messageHandlingOption="Cache"/>
    </handler>
  </before>
  <after>
    <handler handlerName="DuplicateMessage">
      <duplicateHandlerSettings lifeTime="60"
        messageHandlingOption="Cache"/>
    </handler>
    <handler handlerName="SyntacticValidation">
      <syntactValidationSettings
        requestSchema="C:\Program Files\Microsoft EDRA\CS\
          ReferenceImplementation\GlobalBank\Services\Schemas\
          BillPaymentRequest.xsd"
        responseSchema=""/>
    </handler>
    <handler handlerName="ExecutionTimeout"/>
  </after>
</pipeline>
```

Note: The **requestSchema** path information in the preceding code is displayed on multiple lines due to formatting restrictions. If this configuration is copied, you will need to modify the schema path information so that it is all on one line.

Notice that the **serviceActionName** matches the name of the public method shown in the **BillPaymentService** class. The EDAF uses the method name to look for pipeline and business action entries with the same **serviceActionName**. This pipeline is also configured to use a Web service transport, an **inproc** target, **ExecutionTimeout** handler, **SyntacticValidation** handler, **Identity** handler, and a **DuplicateMessage** handler.

The Service Implementation pipeline configuration in `GlobalBankServices.config` is shown in the following code.

```
<pipeline name="BillPaymentImplementationPipeline"
  serviceActionName="PayPendingBill" targetName="businessAction">
  <before>
    <handler handlerName="LoggingHandler"/>
    <handler handlerName="AppInstrumentation"/>
  </before>
  <after>
  </after>
</pipeline>
```

The Service Implementation pipeline uses the **LoggingHandler** and the **AppInstrumentation** handlers, and executes the business action. When the **businessAction** target is invoked, the framework looks in the `GlobalBankServices.config` file for a **businessAction** element with the same **serviceActionName** attribute, which is shown in the following code.

```
<businessAction serviceActionName="PayPendingBill"
  type="Microsoft.ReferenceImplementation.Services.BillPayment.
    BusinessActions.BillPaymentBusinessAction,
    BillPayment.BusinessActions"
  invocationMethod="Serialization"
  compensate="UndoPayPendingBillTransaction"
  validate="true" validationMethod="ValidateBillPayment">
  <request type="Microsoft.ReferenceImplementation.Services.BillPayment.
    MessageDefinitions.BillPaymentRequest,
    BillPayment.MessageDefinitions">
    <method name="PayPendingBill"/>
  </request>
  <response type="Microsoft.ReferenceImplementation.Services.BillPayment.
    MessageDefinitions.BillPaymentResponse,
    BillPayment.MessageDefinitions"/>
</businessAction>
```

Note: The type information shown in the preceding code was broken across multiple lines due to formatting constraints. If this configuration is copied, you will need to modify the type information so that it is all on one line.

The business action class also defines a public method named **UndoPayPendingBillTransaction**, which is defined as the compensate method in the **businessAction** configuration. The following **businessAction** configuration is used to define the compensate method.

```
<businessAction serviceActionName="UndoPayPendingBillTransaction"
    type="Microsoft.ReferenceImplementation.Services.BillPayment.
        BusinessActions.BillPaymentBusinessAction,
        BillPayment.BusinessActions"
    invocationMethod="Serialization">
    <request type="Microsoft.ReferenceImplementation.Services.BillPayment.
        MessageDefinitions.BillPaymentRequest,
        BillPayment.MessageDefinitions">
        <method name="UndoPayPendingBillTransaction"/>
    </request>
</businessAction>
```

Note: The type information shown in the preceding code was broken across multiple lines due to formatting constraints. If this configuration is copied, you will need to modify the type information so that it is all on one line.

This method takes the **BillPaymentRequest** type as an input parameter and does not return a result. Its purpose is to undo changes that were made during a **PayPendingBill** operation.

Handlers

Table 20 lists the handlers this service uses.

Table 20: BillPayment Handlers

Name	Description
ExecutionTimeout	Aborts the current thread if execution time exceeds 300 seconds.
SyntacticValidation	Uses the BillPaymentRequest.xsd schema to validate the request message. Not configured to validate a response message.
Identity	Extracts the UserName from the message header and creates an authenticated GenericPrincipal using the name.
DuplicateMessage	Keeps track of service action requests to prevent duplicate implementations of the same request. If a result is available from a previous request, it will be returned; otherwise, a null value is returned. This is configured to hold a result for 60 seconds.
LoggingHandler	Writes log entries to the GlobalBank_Core database.
ApplInstrumentation	Updates the following Performance Monitor counters: ReferenceArchitecture.TotalRequests ReferenceArchitecture.RequestsPerSecond

Deployment View

The **BillPayment** service is located within the Global Bank system. The deployment diagram in Figure 12 shows the different systems involved.

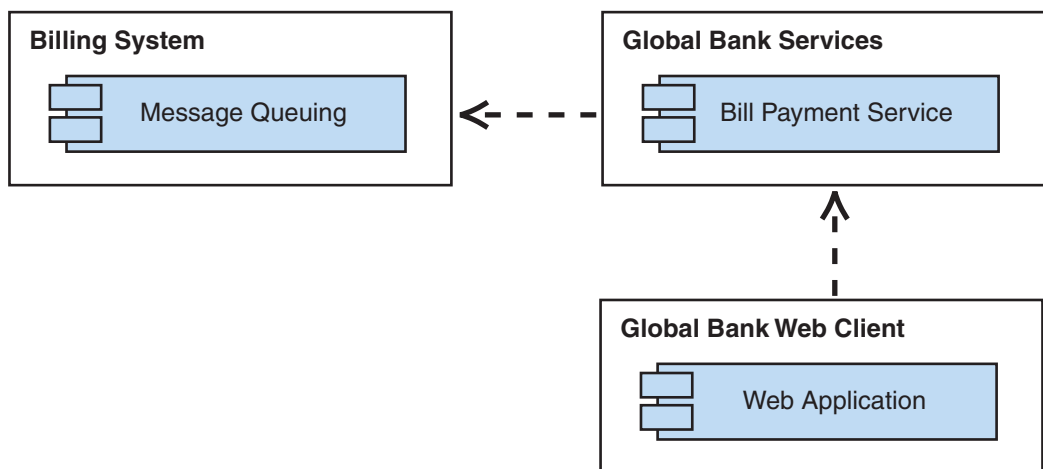


Figure 12

BillPayment service deployment diagram

BillSubscription Service

This service provides actions to subscribe or unsubscribe from bill payment subscriptions.

Logical View

The Enterprise Development Application Framework requires each service to define a Service Interface transport and a business action component. For details about the framework, see the [Enterprise Development Reference Architecture](#) documentation. In addition, the “Funds Transfer Walkthrough” section describes how the framework processes a service request.

The service interface transport used for this service is a Web service named **BillSubscriptionService**, as follows:

BillSubscriptionService
+Credentials : RequestHeader
+SubscribeToPayBill(BillSubscriptionRequest : BillSubscriptionRequest) : BillSubscriptionResponse
+UnsubscribeToPayBill(BillSubscriptionRequest : BillSubscriptionRequest) : BillSubscriptionResponse

The **BillSubscriptionService** class defines two public methods named **SubscribeToPayBills** and **UnsubscribeToPayBills**. Both methods take a **BillSubscriptionRequest** as the request parameter and returns a **BillSubscriptionResponse**. This class also defines a public attribute named **Credentials**, which is a SOAP request header.

The SOAP header should contain the following values:

Name	Value
UserName	Name of the user.
Channel	Identifies the application used to make the request.

The business action class for the **BillSubscriptionService** is named **BillSubscriptionBusinessAction**. This class also defines two methods named **SubscribeToPayBills** and **UnsubscribeToPayBills**, which both take the same parameters as the bill subscription service methods. In addition, both of these actions are very simple as shown in the diagram in Figure 13.

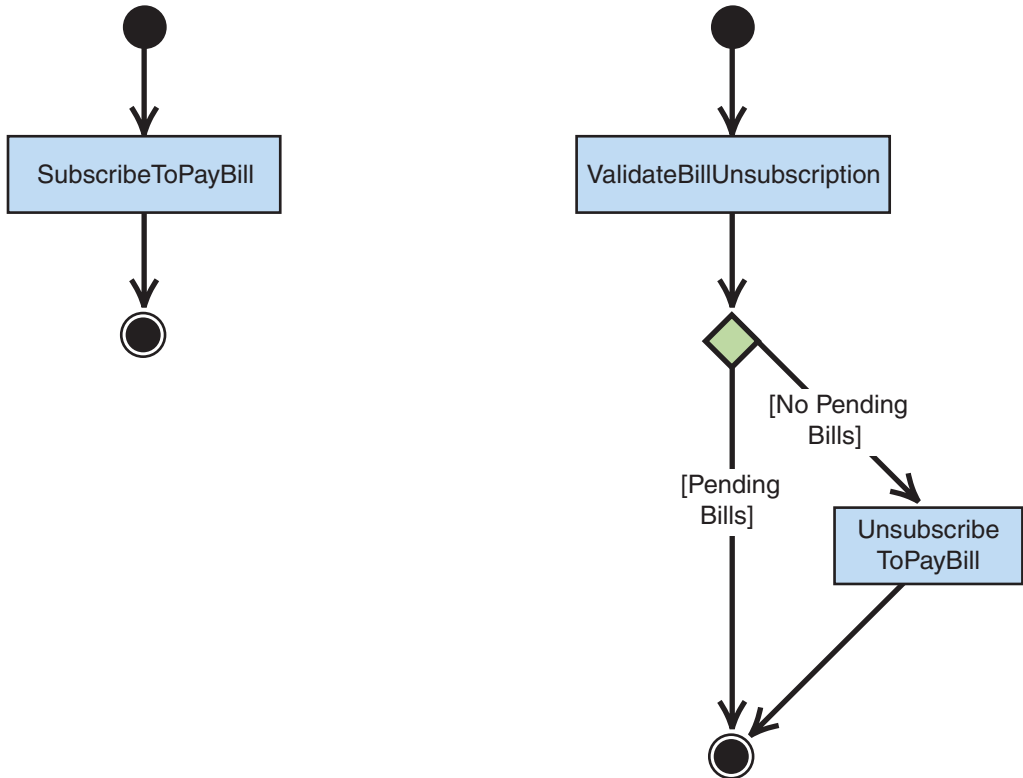


Figure 13

Bill Subscription activity diagrams

From the diagram in Figure 42, you can see that both business actions perform a single operation and then return.

Client Interface

To interact with the service, a client will need to know the structure of messages, the configuration information of this service, and the handlers that it will use.

Messages

This service uses a **BillPaymentRequest** message for the input parameters and returns the result in a **BillPaymentResponse** message.

Request Message — BillSubscriptionRequest

The **BillSubscriptionRequest** message uses a single type that contains the elements listed in Table 21.

Table 21: BillSubscriptionRequest

Element	Type	Description
PayeeIdentifier	Int32	Unique Identifier for the payee that offers bill subscription service.
PayeeName	EntityNameType	Name of the payee for which the bill is being paid.
CustomerExternalIdentifier	ExternalIdentifierType	User account number in the payee system.

Response Message — BillSubscriptionResponse

The **BillSubscriptionResponse** type contains **BillSubscription** information along with the response information, shown in Figure 14.

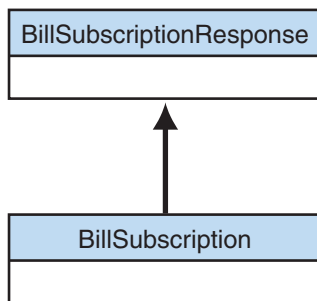


Figure 14

BillSubscriptionResponse

The following tables provide information about the types and element names used to define the response message.

Table 22: BillSubscriptionResponse

Element	Type	Description
UserSubscribedBill	BillSubscription	Contains the bill the user is subscribed to or unsubscribed to.
OperationSuccessful	Boolean	Indicates whether the subscription to the bill or deletion of a current subscription was successful.
FailureReason	String	If the subscription failed contains the reason for the failure.

Table 23: BillSubscription

Element	Type	Description
CustomerExternalIdentifier	String	ID of the customer
AccountActive	Boolean	Flag indicating active state
PayeeId	Int32	Link to payee information
PayeeName	String	Name of the payee

Service Configuration

The Service Interface pipeline configuration for the **SubscribeToPayBill** action in `GlobalBankServices.config` is shown in the following code.

```
<pipeline name="BillSubscriptionPipeline"
  transportName="WebServiceTransport"
  serviceActionName="SubscribeToPayBill"
  targetName="inproc">
  <before>
    <handler handlerName="ExecutionTimeout">
      <timeoutConfiguration timeout="300"/>
    </handler>
    <handler handlerName="SyntacticValidation">
      <syntactValidationSettings
        requestSchema="C:\Program Files\Microsoft EDRA\CS\
          ReferenceImplementation\GlobalBank\Services\Schemas\
          BillSubscriptionRequest.xsd"
        responseSchema=""/>
      </handler>
    <handler handlerName="Identity"/>
    <handler handlerName="DuplicateMessage">
      <duplicateHandlerSettings lifeTime="60"
        messageHandlingOption="Cache"/>
    </handler>
  </before>
  <after>
    <handler handlerName="DuplicateMessage">
      <duplicateHandlerSettings lifeTime="60"
        messageHandlingOption="Cache"/>
    <handler handlerName="SyntacticValidation">
      <syntactValidationSettings
        requestSchema="C:\Program Files\Microsoft EDRA\CS\
          ReferenceImplementation\GlobalBank\Services\Schemas\
          BillSubscriptionRequest.xsd"
        responseSchema=""/>
      </handler>
    <handler handlerName="ExecutionTimeout"/>
  </after>
</pipeline>
```

This pipeline is configured to use a Web service transport, an **inproc** target, **ExecutionTimeout** handler, **SyntacticValidation** handler, **Identity** handler, and a **DuplicateMessage** handler. The Service Implementation pipeline configuration in `GlobalBankServices.config` is shown in the following code.

```
<pipeline name="BillSubscriptionImplementationPipeline"
  serviceActionName="SubscribeToPayBill" targetName="businessAction">
  <before>
    <handler handlerName="LoggingHandler"/>
    <handler handlerName="AppInstrumentation"/>
  </before>
  <after>
  </after>
</pipeline>
```

The Service Implementation pipeline uses the **LoggingHandler** and the **AppInstrumentation** handlers, and executes the business action.

The Service Interface pipeline configuration for the **UnSubscribeToPayBill** in `GlobalBankServices.config` action is shown in the following code.

```
<pipeline name="UnsubscribeToPayBillPipeline"
  transportName="WebServiceTransport"
  serviceActionName="UnsubscribeToPayBill"
  targetName="inproc">
  <before>
    <handler handlerName="ExecutionTimeout">
      <timeoutConfiguration timeout="300"/>
    </handler>
    <handler handlerName="SyntacticValidation">
      <syntactValidationSettings
        requestSchema="C:\Program Files\Microsoft EDRA\CS\
          ReferenceImplementation\GlobalBank\Services\Schemas\
          BillSubscriptionRequest.xsd"
        responseSchema=""/>
    </handler>
    <handler handlerName="Identity"/>
    <handler handlerName="DuplicateMessage">
      <duplicateHandlerSettings lifeTime="60"
        messageHandlingOption="Cache"/>
    </handler>
  </before>
  <after>
    <handler handlerName="DuplicateMessage">
      <duplicateHandlerSettings lifeTime="60"
        messageHandlingOption="Cache"/>
    </handler>
```

(continued)

(continued)

```

        <handler handlerName="SyntacticValidation">
            <syntacticValidationSettings
                requestSchema=""
                responseSchema="C:\Program Files\Microsoft EDRA\CS\
                    ReferenceImplementation\GlobalBank\Services\Schemas\
                    BillSubscriptionRequest.xsd"/>
            </handler>
        <handler handlerName="ExecutionTimeout"/>
    </after>
</pipeline>

```

This pipeline is configured to use a Web service transport, an **inproc** target, **ExecutionTimeout** handler, **SyntacticValidation** handler, **Identity** handler, and a **DuplicateMessage** handler. The Service Implementation pipeline configuration in GlobalBankServices.config is shown in the following code.

```

<pipeline name="UnsubscribeToPayBillImplementationPipeline"
    serviceActionName="UnsubscribeToPayBill" targetName="businessAction">
    <before>
        <handler handlerName="LoggingHandler"/>
        <handler handlerName="AppInstrumentation"/>
    </before>
    <after>
    </after>
</pipeline>

```

The Service Implementation pipeline uses the **LoggingHandler** and the **AppInstrumentation** handlers, and executes the business action. The only difference between the configuration for the **SubscribeToPayBill** action and **UnSubscribeToPayBill** action is the **serviceActionName** because there is a separate pipeline for each service action. When the **businessAction** target is invoked, the framework looks in the GlobalBankServices.config file for a **businessAction** element with the same **serviceActionName** attribute. Both business action configurations are shown in the following code.

```

<businessAction serviceActionName="SubscribeToPayBill"
    type="Microsoft.ReferenceImplementation.Services.
        BillSubscription.BusinessActions.
        BillSubscriptionBusinessAction,
        BillSubscription.BusinessActions"
    invocationMethod="Serialization">
    <request type="Microsoft.ReferenceImplementation.Services.
        BillSubscription.MessageDefinitions.
        BillSubscriptionRequest,
        BillSubscription.MessageDefinitions">
        <method name="SubscribeToPayBill"/>
    </request>

```

(continued)

(continued)

```

        <response type="Microsoft.ReferenceImplementation.Services.
            BillSubscription.MessageDefinitions.
            BillSubscriptionResponse,
            BillSubscription.MessageDefinitions"/>
    </businessAction>
    <businessAction serviceActionName="UnsubscribeToPayBill"
        type="Microsoft.ReferenceImplementation.Services.
            BillSubscription.BusinessActions.
            BillSubscriptionBusinessAction,
            BillSubscription.BusinessActions"
        invocationMethod="Serialization">
        <request type="Microsoft.ReferenceImplementation.Services.
            BillSubscription.MessageDefinitions.
            BillSubscriptionRequest,
            BillSubscription.MessageDefinitions">
            <method name="UnsubscribeToPayBill"/>
        </request>
        <response type="Microsoft.ReferenceImplementation.Services.
            BillSubscription.MessageDefinitions.
            BillSubscriptionResponse,
            BillSubscription.MessageDefinitions"/>
    </businessAction>

```

Handlers

Table 24 lists the handlers this service uses.

Table 24: BillSubscription Handlers

Name	Description
ExecutionTimeout	Aborts the current thread if execution time exceeds 300 seconds.
SyntacticValidation	Uses the BillSubscriptionRequest.xsd schema to validate the request message. Not configured to validate a response message.
Identity	Extracts the UserName from the message header and creates an authenticated GenericPrincipal using the name.
DuplicateMessage	Keeps track of service action requests to prevent duplicate implementations of the same request. If a result is available from a previous request it will be returned, otherwise a null value is returned. This is configured to hold a result for 60 seconds.
LoggingHandler	Writes log entries to the GlobalBank_Core database.
AppInstrumentation	Updates the following Performance Monitor counters: ReferenceArchitecture.TotalRequests ReferenceArchitecture.RequestsPerSecond

Deployment View

The **BillSubscription** service is located within the Global Bank system. The deployment diagram in Figure 15 shows the different systems involved.

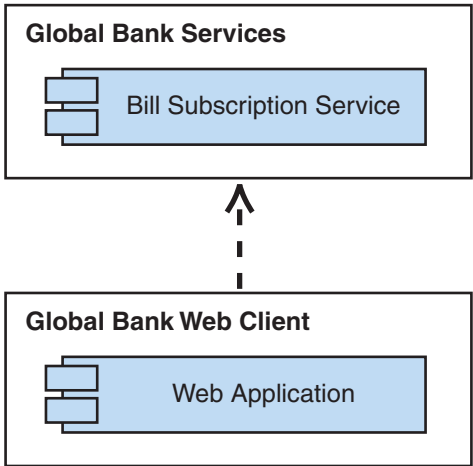


Figure 15
BillSubscription service deployment diagram

FundsTransfer Service

This service is used to transfer funds between accounts.

Logical View

The Enterprise Development Application Framework requires each service to define a Service Interface transport and a business action component. For details about the framework, see the [Enterprise Development Reference Architecture](#) documentation. In addition, the “Funds Transfer Walkthrough” section describes how the framework processes a service request.

The service interface transport used for this service is a Web service named **FundsTransferService**, as follows:

FundsTransferService
+Credentials : RequestHeader
+PerformFundsTransfer(FundsTransferRequest : FundsTransferRequest) : FundsTransferResponse

The **FundsTransferService** class defines a public method named **PerformFundsTransfer**, which takes a **FundsTransferRequest** as the request parameter and returns a **FundsTransferResponse**. This class also defines a public attribute named **Credentials**, which is a SOAP request header.

The SOAP header should contain the following values:

Name	Value
UserName	Name of the user.
Channel	Identifies the application used to make the request.

The business action class for the **FundsTransferService** class is named **FundsTransferBusinessAction**. This class also defines two public methods named **PerformFundsTransfer** and **ValidateSourceAccount**. The business action class is responsible for validating account information and transferring funds as shown in the activity diagram in Figure 16.

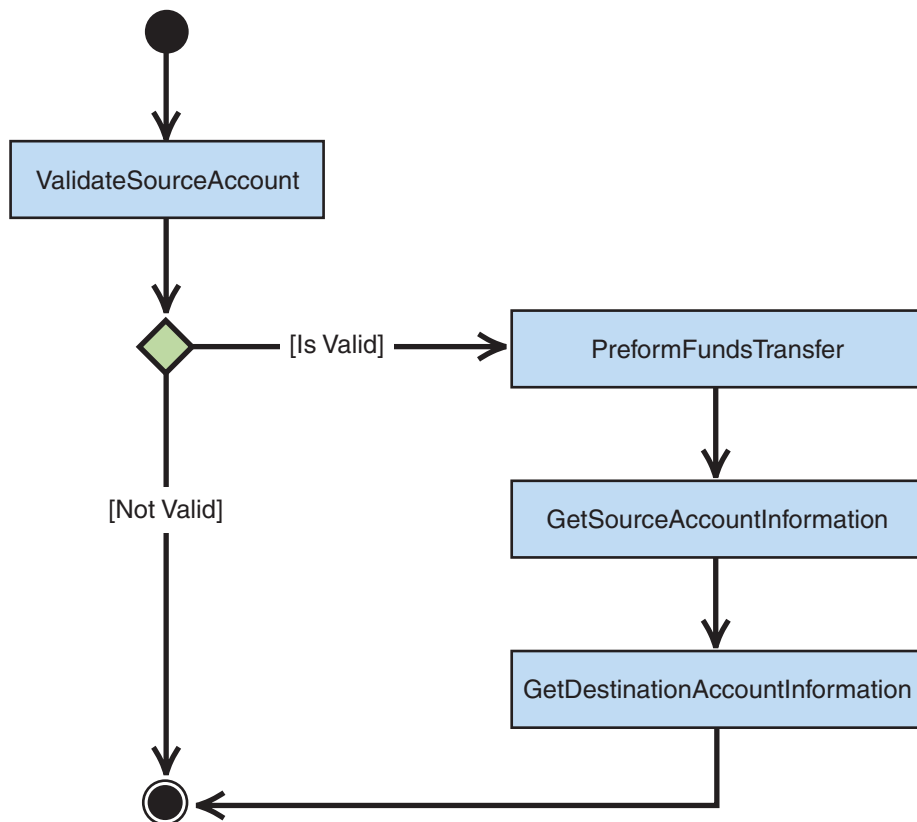


Figure 16

Funds Transfer activity diagram

As you can see in the preceding diagram, the first action is to validate the source account. If the account is valid, a transfer is performed, and account information for both the source and destination accounts are retrieved.

Client Interface

To interact with the service, a client will need to know the structure of messages, the configuration information of this service, and the handlers that it will use.

Messages

This service uses a **FundsTransferRequest** message for the input parameters and returns the result in a **FundsTransferResponse** message.

Request Message — FundsTransferRequest

The **FundsTransferRequest** message uses a single type that contains the elements listed in Table 25.

Table 25: FundsTransferRequest

Element	Type	Description
SourceAccountNumber	AccountNumberType	The account number identifying the source account for the bill payment.
DestinationAccountNumber	AccountNumberType	The account number identifying the destination account for the funds transfer.
Amount	Decimal	Amount to transfer.
Description	DescriptionType	Description provided by the user during funds transfer.
EffectiveOn	DateTime	Date on which the transfer will be completed. This is set to the current date when the transfer was initiated.

Response Message — FundsTransferResponse

The **FundsTransferResponse** type contains two different elements with **CashAccount** information along with the response information, as shown in Figure 17.

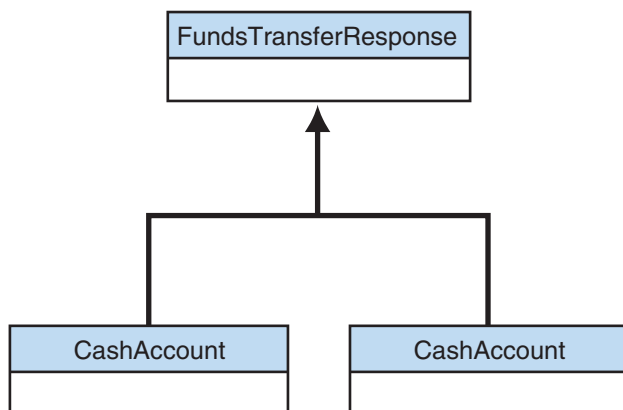


Figure 17

FundsTransferResponse

The following tables provide information about the types and element names used to define the response message.

Table 26: FundsTransferResponse

Element	Type	Description
SourceAccount	CashAccount	The source account for the funds transfer.
DestinationAccount	CashAccount	The destination account for the funds transfer.
TransferSuccessful	Boolean	Indicates whether the transfer was successful.
FailureReason	String	If the transfer failed contains the reason for failure.
ReferenceCode	Int32	Reference code for the transaction that the user can use to track the transaction.

Table 27: CashAccount

Element	Type	Description
AccountNumber	String	Account number uniquely identifying the account.
ProductTypeId	Int32	The integral value maps to indicates the type of the product, such as a savings account or checking account.
Balance	Decimal	The balance amount in the account.
AccountType	String	Type of account, such as a checking account or savings account.

Service Configuration

The Service Interface pipeline configuration in GlobalBankServices.config is shown in the following code.

```
<pipeline name="FundsTransferPipeline"
  transportName="WebServiceTransport"
  serviceActionName="PerformFundsTransfer"
  targetName="inproc">
  <before>
    <handler handlerName="ExecutionTimeout">
      <timeoutConfiguration timeout="300"/>
    </handler>
    <handler handlerName="SyntacticValidation">
      <syntactValidationSettings
        requestSchema="C:\Program Files\Microsoft EDRA\CS\
          ReferenceImplementation\GlobalBank\Services\Schemas\
          FundsTransferRequest.xsd"
        responseSchema=""/>
      </handler>
    <handler handlerName="Identity"/>
    <handler handlerName="DuplicateMessage">
      <duplicateHandlerSettings lifeTime="60"
        messageHandlingOption="Cache"/>
    </handler>
  </before>
  <after>
    <handler handlerName="DuplicateMessage">
      <duplicateHandlerSettings lifeTime="60"
        messageHandlingOption="Cache"/>
    </handler>
    <handler handlerName="SyntacticValidation">
      <syntactValidationSettings
        requestSchema="C:\Program Files\Microsoft EDRA\CS\
          ReferenceImplementation\GlobalBank\Services\Schemas\
          FundsTransferRequest.xsd"
        responseSchema=""/>
      </handler>
    <handler handlerName="ExecutionTimeout"/>
  </after>
</pipeline>
```

Note: The **requestSchema** path information in the preceding code is displayed on multiple lines due to formatting restrictions. If this configuration is copied, you will need to modify the schema path information so that it is all on one line.

Notice that the **serviceActionName** matches the name of the public method shown in the **FundsTransferService** class. The EDAF uses the method name to look for pipeline and business action entries with the same **serviceActionName**. This pipeline is also configured to use a Web service transport, an **inproc** target, an **ExecutionTimeout** handler, **SyntacticValidation** handler, **Identity** handler, and a **DuplicateMessage** handler.

The Service Implementation pipeline configuration in `GlobalBankServices.config` is shown in the following code.

```
<pipeline name="FundsTransferImplementationPipeline"
  serviceActionName="PerformFundsTransfer"
  targetName="businessAction">
  <before>
    <handler handlerName="LoggingHandler"/>
    <handler handlerName="AppInstrumentation"/>
  </before>
  <after>
  </after>
</pipeline>
```

The Service Implementation pipeline uses the **LoggingHandler** and the **AppInstrumentation** handlers, and executes the business action. When the **businessAction** target is invoked, the framework looks in the `GlobalBankServices.config` file for a **businessAction** element with the same **serviceActionName** attribute, which is shown in the following code.

```
<businessAction serviceActionName="PerformFundsTransfer"
  type="Microsoft.ReferenceImplementation.Services.FundsTransfer.
    BusinessActions.FundsTransferBusinessAction,
    FundsTransfer.BusinessActions"
  invocationMethod="Serialization" validate="true"
  validationMethod="ValidateSourceAccount">
  <request type="Microsoft.ReferenceImplementation.Services.FundsTransfer.
    MessageDefinitions.FundsTransferRequest,
    FundsTransfer.MessageDefinitions">
    <method name="PerformFundsTransfer"/>
  </request>
  <response type="Microsoft.ReferenceImplementation.Services.FundsTransfer.
    MessageDefinitions.FundsTransferResponse,
    FundsTransfer.MessageDefinitions"/>
</businessAction>
```

Note: The type information shown in the preceding code was broken across multiple lines due to formatting constraints. If this configuration is copied, you will need to modify the type information so that it is all on one line.

Handlers

Table 28 lists the handlers this service uses.

Table 28: FundsTransfer Handlers

Name	Description
ExecutionTimeout	Aborts the current thread if execution time exceeds 300 seconds.
SyntacticValidation	Uses the FundsTransferRequest.xsd schema to validate the request message. Not configured to validate a response message.
Identity	Extracts the UserName from the message header and creates an authenticated GenericPrincipal using the name.
DuplicateMessage	Keeps track of service action requests to prevent duplicate implementations of the same request. If a result is available from a previous request, it will be returned; otherwise, a null value is returned. This is configured to hold a result for 60 seconds.
LoggingHandler	Writes log entries to the GlobalBank_Core database.
ApplInstrumentation	Updates the following Performance Monitor counters: ReferenceArchitecture.TotalRequests ReferenceArchitecture.RequestsPerSecond

Deployment View

The **FundsTransfer** service is located within the Global Bank system. The deployment diagram in Figure 18 shows the different systems involved.

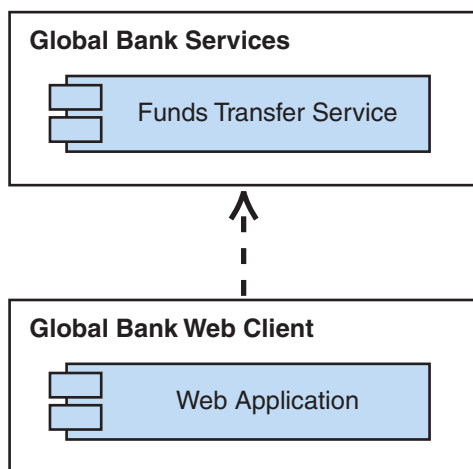


Figure 18

FundsTransfer service deployment diagram

TransactionLog Service

This service retrieves transaction information from database logs. This information can be used to track transactions performed by a user.

Logical View

The Enterprise Development Application Framework requires each service to define a Service Interface transport and a business action component. For details about the framework, see the [Enterprise Development Reference Architecture](#) documentation. In addition, the “Funds Transfer Walkthrough” section describes how the framework processes a service request.

The Service Interface transport used for this service is a Web service named **TransactionLogService**, as follows:

TransactionLogService	
+Credentials :	RequestHeader
+GetCustomerTransactionLog(TransactionLogReportRequest :	
TransactionLogReportRequest) :	TransactionLogReportResponse

The **TransactionLogService** class defines a public method named **GetCustomerTransactionLog**, which takes a **TransactionLogReportRequest** as the request parameter and returns a **TransactionLogReportResponse**. This class also defines a public attribute named **Credentials**, which is a SOAP request header.

The SOAP header should contain the following values:

Name	Value
UserName	Name of the user.
Channel	Identifies the application used to make the request.

The business action class for the **TransactionLogService** is named **TransactionLogBusinessAction**. This class defines two public methods named **GetCustomerTransactionLog** and **ValidateInputDateRange**. The business action class is responsible for validating date ranges and retrieving the transaction log as shown in the activity diagram in Figure 19.

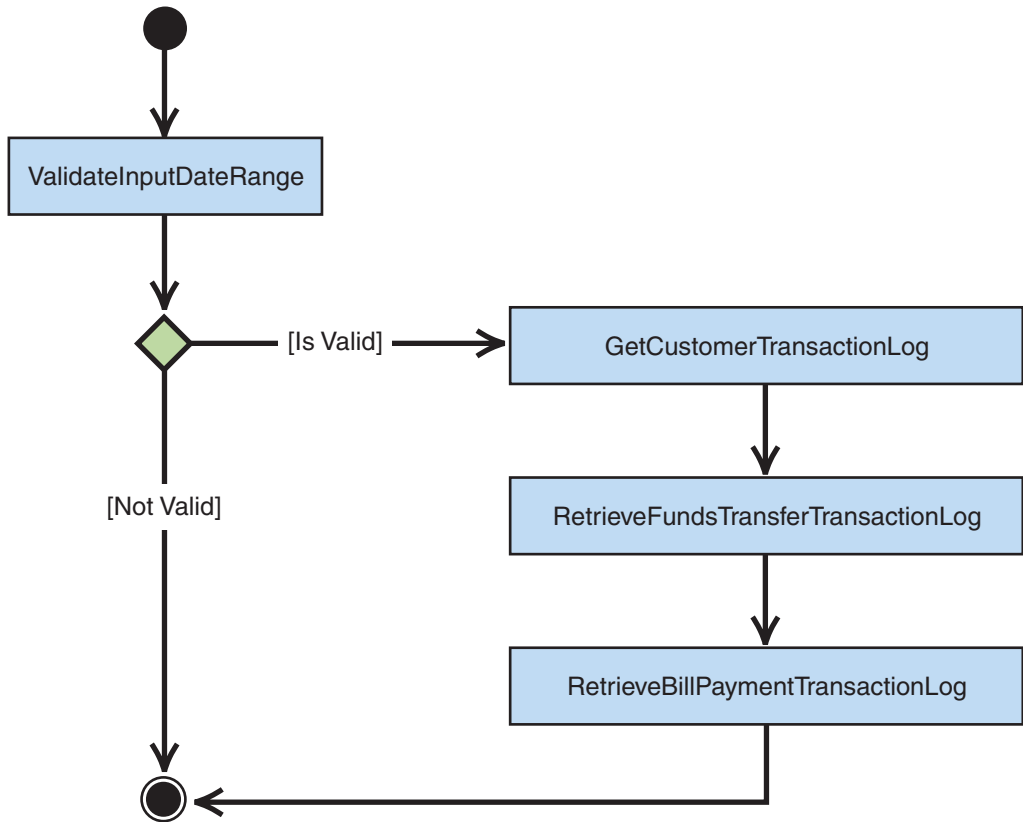


Figure 19
Transfer Log activity diagram

Client Interface

To interact with the service, a client will need to know the structure of messages, the configuration information of this service, and the handlers that it will use.

Messages

This service uses a **TransactionLogReportRequest** message for the input parameters and returns the result in a **TransactionLogReportResponse** message.

Request Message — TransactionLogReportRequest

The **TransactionLogReportRequest** message uses a single type that contains the elements listed in Table 29.

Table 29: TransactionLogReportRequest

Element	Type	Description
StartDate	DateTime	The start date for the transaction log report.
EndDate	DateTime	The end date for the transaction log report.

Response Message — TransactionLogReportResponse

The **TransactionLogReportResponse** type contains two different elements with **BillPayment** and **FundsTransfer** information along with the response information, as shown in Figure 20.

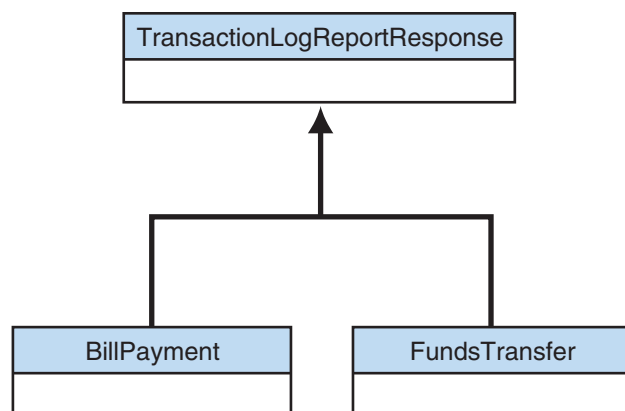


Figure 20

TransactionLogReportResponse

The following tables provide information about the types and element names used to define the response message.

Table 30: TransactionLogReportResponse

Element	Type	Description
TransactionLogReportFundsTransfer Collection	FundsTransferCollection	Includes all of the funds transfers initiated during the specified date range.
TransactionLogReportBillPayment Collection	BillPaymentCollection	Includes all the bill payments initiated during the date range.

Table 31: BillPayment

Element	Type	Description
ReferenceCode	String	Reference number generated for the transaction.
BillIdInBillingSystem	String	External identifier for the bill in the payee system.
SourceAccountId	String	Identifier for the account from which the bill is being paid.
Amount	Decimal	Amount of the bill.
DateEffectiveOn	DateTime	The date on which the bill payment is completed. This version defaults to the current date.
Executed	Boolean	Indicates whether the bill payment was completed or not.
PayeeName	String	Indicates the payee name.
CustomerIdInBillingSystem	String	Customer account number in the external system.

Table 32: FundsTransfer

Element	Type	Description
ReferenceCode	String	Reference number generated for the transaction.
SourceAccountId	String	Unique identifier for the source account in the funds transfer.
DestinationAccountId	String	Unique identifier for the destination account in the funds transfer.
AmountToTransfer	Decimal	Amount being transferred.
DateEffectiveOn	DateTime	Date on which the transfer is done. This will always default to the current date in this version.
Description	String	The description specified by the user when initiating the funds transfer.
Executed	Boolean	Indicates whether the transfer was completed or not.

Service Configuration

The Service Interface pipeline configuration in GlobalBankServices.config is shown in the following code.

```
<pipeline name="TransactionLogPipeline"
  transportName="WebServiceTransport"
  serviceActionName="GetCustomerTransactionLog"
  targetName="inproc">
  <before>
    <handler handlerName="ExecutionTimeout">
      <timeoutConfiguration timeout="300"/>
    </handler>
    <handler handlerName="SyntacticValidation">
      <syntactValidationSettings
        requestSchema="C:\Program Files\Microsoft EDRA\CS\
          ReferenceImplementation\GlobalBank\Services\Schemas\
          TransactionLogRequest.xsd"
        responseSchema=""/>
      </handler>
    <handler handlerName="Identity"/>
  </before>
  <after>
    <handler handlerName="SyntacticValidation">
      <syntactValidationSettings
        requestSchema="C:\Program Files\Microsoft EDRA\CS\
          ReferenceImplementation\GlobalBank\Services\Schemas\
          TransactionLogRequest.xsd"
        responseSchema=""/>
      </handler>
    <handler handlerName="ExecutionTimeout"/>
  </after>
</pipeline>
```

Note: The **requestSchema** path information in the preceding code is displayed on multiple lines due to formatting restrictions. If this configuration is copied, you will need to modify the schema path information so that it is all on one line.

Notice that the **serviceActionName** matches the name of the public method shown in the **TransactionLogService** class. The EDAF uses the method name to look for pipeline and business action entries with the same **serviceActionName**. This pipeline is also configured to use a Web service transport, an **inproc** target, an **ExecutionTimeout** handler, **SyntacticValidation** handler, and **Identity** handler.

The Service Implementation pipeline configuration in GlobalBankServices.config is shown in the following code.

```
<pipeline name="TransactionLogImplementationPipeline"
  serviceActionName="GetCustomerTransactionLog"
  targetName="businessAction">
  <before>
    <handler handlerName="AppInstrumentation"/>
  </before>
  <after>
  </after>
</pipeline>
```

The Service Implementation pipeline uses the **AppInstrumentation** handler and executes the business action. When the **businessAction** target is invoked, the framework looks in the GlobalBankServices.config file for a **businessAction** element with the same **serviceActionName** attribute, which is shown in the following code.

```
<businessAction serviceActionName="GetCustomerTransactionLog"
  type="Microsoft.ReferenceImplementation.Services.
    TransactionLog.BusinessActions.
    TransactionLogBusinessAction,
    TransactionLog.BusinessActions"
  invocationMethod="Serialization"
  validate="true" validationMethod="ValidateInputDateRange">
  <request type="Microsoft.ReferenceImplementation.Services.TransactionLog.
    MessageDefinitions.TransactionLogReportRequest,
    TransactionLog.MessageDefinitions">
    <method name="GetCustomerTransactionLog"/>
  </request>
  <response type="Microsoft.ReferenceImplementation.Services.TransactionLog.
    MessageDefinitions.TransactionLogReportResponse,
    TransactionLog.MessageDefinitions"/>
</businessAction>
```

Note: The type information shown in the preceding code was broken across multiple lines due to formatting constraints. If this configuration is copied, you will need to modify the type information so that it is all on one line.

Handlers

Table 33 lists the handlers this service uses.

Table 33: TransactionLog Handlers

Name	Description
ExecutionTimeout	Aborts the current thread if execution time exceeds 300 seconds.
SyntacticValidation	Uses the TransactionLogRequest.xsd schema to validate the request message. Not configured to validate a response message.
Identity	Extracts the UserName from the message header and creates an authenticated GenericPrincipal using the name.
ApplInstrumentation	Updates the following Performance Monitor counters: ReferenceArchitecture.TotalRequests ReferenceArchitecture.RequestsPerSecond

Deployment View

The **TransactionLog** service is located within the Global Bank system. The deployment diagram in Figure 21 shows the different systems involved.

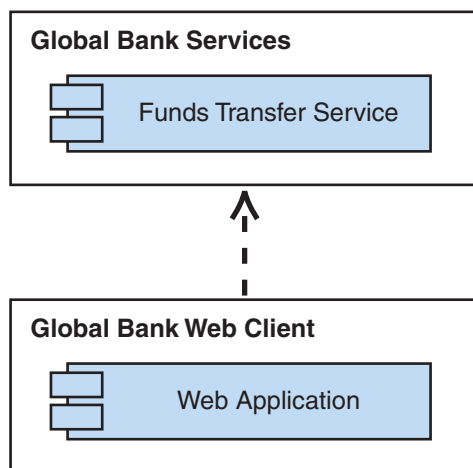


Figure 21

TransactionLog service deployment diagram

Summary

The Enterprise Development Reference Implementation applies *patterns & practices* guidance and demonstrates the use of the Enterprise Development Application Framework. This document outlined the design objectives and principles for the Global Bank Internet banking application Presentation tier and the EDRI services. For additional information, see “Appendix A — Inside the Enterprise Development Application Framework” and “Appendix B — Exploring the EDAF Using the Bill Payment Use Case.”

More Information

For information about Visual Studio.NET, see the Microsoft Visual Studio Developer Center at <http://msdn.microsoft.com/vstudio/>.

For information about ASP.NET, see the Microsoft ASP.NET Developer Center at <http://msdn.microsoft.com/asp.net/>.

For information about Enterprise Services, see “Understanding Enterprise Services (COM+) in .NET,” at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/entserv.asp>.

For information about BizTalk Server, see <http://msdn.microsoft.com/library/default.asp?url=/nhp/default.asp?contentid=28000399>.

For information on Web Service Enhancements (WSE), see “Programming with Web Services Enhancements 2.0” at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwse/html/programwse2.asp>.

For information about Indigo, see Microsoft “Indigo” Frequently Asked Questions at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnlong/html/indigofaq1.asp>.

For information about designing distributed applications, see “Application Architecture for .NET: Designing Applications and Services,” at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/distapp.asp>.

To interact with the Enterprise Development Reference Architecture Community, see the GotDotNet workspace, at <http://go.microsoft.com/fwlink/?LinkId=31528>.

To interact with the Enterprise Development Reference Architecture Community, you can also use the Wiki at <http://go.microsoft.com/fwlink/?LinkId=31530>.

To interact with the Enterprise Development Reference Implementation Community, you can also use the Wiki at <http://go.microsoft.com/fwlink/?LinkId=31531>.

For information about online banking application response time statistics; see www.keynote.com or www.gomez.com.

Improving Web Application Security: Threats and Countermeasures, Redmond: Microsoft Press, 2003, ISBN: 0735618429. Also available on MSDN at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/ThreatCounter.asp>.

Building Secure Microsoft ASP.Net Applications — Authentication, Authorization and Secure Communication, Microsoft Press, ISBN: 0735618909, available at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/secnetlpMSDN.asp>.

Improving .NET Application Performance and Scalability, Microsoft Press, ISBN: 0735618518, available at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/scalenet.asp>.

Frank Buschmann et al., *Pattern-Oriented Software Architecture, Volume 1, A System of Patterns*. Chichester, England: John Wiley & Sons, 1996

Hofmeister, Christine, Robert Nord, and Dilip Soni. *Applied Software Architecture*. Reading Massachusetts: Addison-Wesley, 1999

Enterprise Solution Patterns Using Microsoft .NET, Redmond: Microsoft Press, 2003, ISBN: 0735618399. Also available on MSDN at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpatterns/html/Esp.asp>.

For information about the Model-View-Controller pattern in *Enterprise Solution Patterns Using Microsoft .NET* at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpatterns/html/DesMVC.asp>.

For information about the Page Controller pattern in *Enterprise Solution Patterns Using Microsoft .NET* at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpatterns/html/DesPageController.asp>.

For information about the Service Interface Pattern in *Enterprise Solution Patterns Using Microsoft .NET* at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpatterns/html/DesServiceInterface.asp>.

For information about the Front Controller pattern in *Enterprise Solution Patterns Using Microsoft .NET* at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpatterns/html/DesFrontController.asp>.

For information about the Service-Oriented Integration Pattern, see *Integration Patterns* at <http://msdn.microsoft.com/architecture/patterns/default.aspx?pull=/library/en-us/dnpag/html/archserviceorientedintegration.asp>.

4

Installation

Introduction

This section describes how to install the Enterprise Development Reference Implementation (EDRI). Before you install the EDRI, you should have the Enterprise Development Application Framework (EDAF) installed. For details, see the “Before You Begin” section.

Platform Prerequisites

To install and use the EDRI, you need to ensure that your system meets the following minimum software requirements:

- Microsoft® Windows® 2000, Windows XP, or Windows Server™ 2003 operating system

Note: If you use Windows Server 2003, verify that ASP.NET is enabled. You can do this from Add/Remove Programs in Control Panel. Click **Add/Remove Windows Components**, select **Application Server**, and then click the **Details** button. Verify that the **ASP.NET** check box is selected.

- Microsoft Internet Information Server (IIS) version 5.1 or later
- Microsoft .NET Framework version 1.1
- Microsoft SQL Server™ 2000 with SP3a
- Microsoft Windows Message Queuing (also known as MSMQ)
- [Microsoft Enterprise Instrumentation Framework \(EIF\)](#)
- Microsoft WMI (Windows Management Instrumentation)
- Enterprise Development Application Framework (EDAF) version 1.1
- Microsoft Visual Studio® .NET 2003 development system, Enterprise Architect edition or Enterprise Developer edition

Note: The EDAF requires one of the enterprise editions of Microsoft Visual Studio .NET 2003 because these editions have enterprise template support.

Before You Begin

If you have not already installed the EDAF, you can download it from the [community workspace](#).

► To install the EDAF

1. In the folder where you downloaded the .msi file, double-click **Enterprise Development Reference Architecture.msi**.
2. In the **Welcome to the Enterprise Development Reference Architecture Setup Wizard** dialog box, read the provided information, and then click **Next** if you agree or click **Cancel** to exit the installation.
3. Review the terms of the End User License Agreement. If you agree to the terms and conditions, select **I Agree**, and then click **Next**. If you do not agree, click **Cancel** to exit the installation.
4. In the **Enterprise Development Reference Architecture** dialog box, select the appropriate SNK file generation option, and then click **Next**. For a development environment, the default option is appropriate.
5. In the **Select Installation Folder** dialog box, change the default installation folder, if desired, select **Everyone** or **Just Me**, and then click **Next**. The default installation location is C:\Program Files\Microsoft EDRA.
6. In the **Confirm Installation** dialog box, read the provided information. Click **Next** to install the Enterprise Development Reference Architecture, click **Back** to modify the settings, or click **Cancel** to exit the installation.
7. In the **Installation Complete** dialog box, read the provided information, and then click **Close** to complete the installation.

Note: After you install the EDAF, you can access the EDAF documentation. On the taskbar, click **Start**, point to **Programs**, and then point to **Microsoft patterns & practices**. You should see the icon for the *patterns & practices* Web site and the **Enterprise Development Reference Architecture** menu. There are several options when you point to **Enterprise Development Reference Architecture**. One of the options is **Enterprise Development Application Framework Documentation**. This contains supporting documentation for the EDAF and guidance on how to build your own solution using the EDAF.

Before you install the EDRI, make sure the security settings for SQL Server are correct. The SQL Server must have its logon security mode set to SQL Server and Windows authentication. By default, the setup scripts use SQL Server authentication.

► **To check the SQL Server authentication settings**

1. On the taskbar, click **Start**, point to **Programs**, point to **Microsoft SQL Server**, and then click **Enterprise Manager**.
2. In the SQL Server Enterprise Manager window, expand **Microsoft SQL Servers**, expand **SQL Server Group**, right-click your SQL Server instance, and then click **Properties**.
3. In the **SQL Server Properties (Configure)** dialog box, click the **Security** tab. In the **Security** section, select the **SQL Server and Windows** option under **Authentication**.

Note: You can modify the EDRI configuration files for Global Bank to access SQL Server using Windows Integrated Security. However, you must set up the <Machine Name>\ASPNET account in Windows XP. For Windows 2003, use the Windows NT Authority\Network Service account as a user on SQL Server and grant the appropriate permissions. For information, see [“Accessing SQL Server Using Windows Integrated Security.”](#)

You also need to ensure the VSWebCache folder is cleared of any Web cache folders from earlier installations of the framework. If a compile is performed with incorrect configuration path settings in the configuration files, these incorrect settings are cached, and Visual Studio .NET continues to reference the invalid paths.

► **To clear the VSWebCache**

- Delete the following folders from
C:\Documents and Settings\<login account>\VSWebCache\<machine name>:
 - AccountStatementService
 - AuthenticationService
 - BillPaymentService
 - BillSubscriptionService
 - ExternalInvestmentFundSystemService
 - FundsTransferService
 - GlobalBank
 - TransactionLogService

Installing the Enterprise Development Reference Implementation

This section outlines the steps you should follow to install the EDRI.

Note: Before attempting to install the EDRI, you must install the EDAF. For details, see the “Before You Begin” section.

► To install the EDRI

1. In the folder where you downloaded the .msi file, double-click **Enterprise Development Reference Implementation.msi**.
2. In the **Welcome to the Enterprise Development Reference Implementation Setup Wizard** dialog box, read the provided information, and then click **Next** if you agree or click **Cancel** to exit the installation.
3. Review the terms of the End User License Agreement. If you agree to the terms and conditions, select **I Agree**, and then click **Next**. If you do not agree, click **Cancel** to exit the installation.
4. In the **Confirm Installation** dialog box, select **Next** to install the reference implementation or click **Cancel** to exit the installation.
5. In the **Installation Complete** dialog box, read the provided information, and then click **Close** to complete this portion of the installation. The default installation location is the location where the EDRA is installed. For example, C:\Program Files\Microsoft EDRA\CS\ReferenceImplementation.

Verifying the Installation

You should now verify that you have the correct menu items set up. On the taskbar, click **Start**, point to **Programs**, and then point to **Microsoft patterns & practices**. You should see the icon for the *patterns & practices* Web site, the menu for the **Enterprise Development Reference Architecture**, and the **Enterprise Development Reference Implementation** menu. There are several options when you point to **Enterprise Development Reference Implementation**:

- **Build and Deploy Reference Implementation.wsf**. This is the setup script for the EDRI.
- **GlobalBank C#**. This is the solution file for the EDRI.

- **GlobalBank Community Workspace.** The *pattern & practices* team has been using a community workspace as a way for the community members to review the EDRI, and to provide feedback. We encourage you to join the community at our workspace, where you can ask questions, get answers, share your ideas, and provide input for future releases.
- **GlobalBank Wiki.** This is the Web site for the Channel 9 EDRI Discussion Forum. This provides a space for you to discuss the EDRI, get the latest information, and share your suggestions with the development team.

Setting Up and Running the EDRI

This section describes how to set up and run the Enterprise Development Reference Implementation solution. If you encounter any problems setting up the EDRI, see the “Troubleshooting Set Up” section.

► To set up the EDRI

1. On the taskbar, click **Start**, point to **Programs**, point to **Microsoft patterns & practices**, point to **Enterprise Development Reference Implementation**, and then click **Build and Deploy Reference Implementation.wsf**. This script builds and deploys the code.
2. When the script runs, a pop-up window prompts you to confirm whether the software prerequisites are installed. Click **Yes** if you have verified this to be true and want to continue the installation. Otherwise, click **No**.
3. The script prompts you to enter the SQL Server instance name. The default instance name is **(local)**. If the SQL Server database you are using is Microsoft SQL Server Desktop Engine (MSDE), you need to enter the instance name, instead of (local), in the text box in the following format: *server_name\instance_name*. In most cases, this is the instance name of the database, such as the machine name.
4. When the script runs, you will see the following output to the Setup console that launches.

```
Starting Enterprise Development Reference Implementation setup and deploy...
Starting Enterprise Development Reference Architecture setup...
Compiling Services Reference Architecture...
Setting up Business Event Handler...
Setting up Transaction Handler...
Setting up Performance Counters...
Setting up Event Log Source...
Setting up Logging Application Block...
Starting database setup...
Starting compilation of the Enterprise Development Reference Implementation
solution...
Starting config files update...
Starting Transport config files update...
Starting Web Application config file update...
```

(continued)

(continued)

```
Starting deployment of the binaries...
Creating the setup completed flag file...
Setup Completed.
```

Check the Setup.log file for more details

Script finished. Press [ENTER] to close

5. Using the Visual Studio .NET command prompt, go to the <Installation Location> \Microsoft EDRA\CS\ReferenceImplementation directory and run the CreateDatabaseAndPopulateDataForReferenceImplementation.cmd script with **debug** or **release** as the parameter. This script sets up four databases and creates the appropriate accounts for accessing each database. For example, run the script with the following line.

```
CreateDatabaseAndPopulateDataForReferenceImplementation.cmd debug
```

When the script runs, you will see the following output to the Setup console that launches.

```
Copying the UserNamePasswordUtility files to the local folder...
    1 file(s) copied.
    1 file(s) copied.
Creating the GlobalBank_Core database...
Creating the GlobalBank_CreditCard database...
Creating the GlobalBank_CumulativeDeposit database...
Creating the GlobalBank_InvestmentFund database...
Populating database with data...
Populating database with base data...
Populating database with customer data...
Updating User Name and Password...
Removing the UserNamePasswordUtility files from the folder...

Script execution is complete!
```

► To run the EDRI

1. Run the Web application using the following URL:

<http://localhost/GlobalBank/Default.aspx>

Make sure to use the right case when specifying the address in the browser. Global Bank enforces tighter security by restricting access (using the cookie path attribute) to issued cookies, to those originating with the application name "GlobalBank."

Note: When you load the EDRI Web site, it is slower the first time you access the site. This is due to the ASP.NET initialization that occurs the first time. Subsequent loadings will be much faster.

2. Use the following accounts to access Global Bank:

Online ID	Password
11111111	1111
22222222	2222
33333333	3333
44444444	4444
55555555	5555

3. You can open the GlobalBank.sln solution file in Visual Studio .NET 2003 to look at the code. On the taskbar, click **Start**, point to **Programs**, point to **Microsoft patterns & practices**, point to **Enterprise Development Reference Implementation**, and then click **GlobalBank C#**. Alternatively, you can navigate to the <Installation Location>\Microsoft EDRA\CS \ReferenceImplementation directory, and then double-click **GlobalBank.sln**.

Note: By default, the code is built in the debug configuration so that you can debug the code when troubleshooting.

Using the Various Dispatching Transports

An important benefit from using the Enterprise Development Application Framework is that each service's service interface can be separated from its service implementation using four different transports. This separation can be used to add an additional layer of security between the service interface and service implementation, and it can also be used to increase scalability and resiliency of a service. A brief introduction to the EDAF can be found in "Appendix A: Inside the Enterprise Development Application Framework" of the EDRI documentation. For more information about the EDAF, on the taskbar, click **Start**, point to **Programs**, and point to **Microsoft patterns & practices**, point to **Enterprise Development Reference Architecture**, and click on **Enterprise Development Application Framework Documentation**.

By default, the **targetName** attribute for the service interface pipeline is set to **inproc** in the GlobalBankServices.config file. The **inproc** dispatching transport is used to run the service implementation in the same host process as the service interface.

You may also set DCOM, Web service, and Message Queuing as the dispatching transport. This allows you to deploy an additional firewall between the service interface and the service implementation and to use one of the dispatching transports to communicate between the service interface and service implementation.

This section shows you how to modify the default service implementation from inproc to DCOM, Web service, and Message Queuing dispatching transport.

► **To set the DCOM dispatching transport**

1. Open the GlobalBankServices.config file in the <Installation Location>\Microsoft EDRA\CS\ReferenceImplementation\GlobalBank\Services directory. Change the **targetName** attribute to **dcom** for the service interface pipeline being tested, and then save the configuration file.
2. Run the Setup.wsf script from the <Installation Location>\Microsoft EDRA\CS\ReferenceArchitecture\Template\CS directory. From the taskbar, click **Start**, point to **Programs**, point to **Microsoft patterns & practices**, point to **Enterprise Development Reference Architecture**, point to **Starter Templates**, point to **C#**, and then click **Setup.wsf**.
3. Run the Web application using the following URL:
<http://localhost/GlobalBank/Default.aspx>

► **To set the Web service dispatching transport**

1. Open the GlobalBankServices.config file in the <Installation Location>\Microsoft EDRA\CS\ReferenceImplementation\GlobalBank\Services directory. Change the **targetName** attribute to **webservice** for the service interface pipeline being tested, and then save the configuration file.
2. Run the Web application using the following URL:
<http://localhost/GlobalBank/Default.aspx>

► **To set the Message Queuing dispatching transport**

1. Open the GlobalBankServices.config file in the <Installation Location>\Microsoft EDRA\CS\ReferenceImplementation\GlobalBank\Services directory. Change the **targetName** attribute to **msmq** for the service interface pipeline being tested, and then save the configuration file.
2. Run the Setup.wsf script from the <Installation Location>\Microsoft EDRA\CS\ReferenceArchitecture\Template\CS directory. From the taskbar, click **Start**, point to **Programs**, point to **Microsoft patterns & practices**, point to **Enterprise Development Reference Architecture**, point to **Starter Templates**, point to **C#**, and then click **Setup.wsf**.
3. Run the Deploy Reference Implementation.wsf script from the <Installation Location>\Microsoft EDRA\CS\ReferenceImplementation directory.

Note: If you are already using the DCOM dispatching transport and it is running then the Deploy Reference Implementation.wsf script will fail and you will get an “Access denied” error. Before executing the script, you should stop running the DCOM component to resolve the error.

4. Run TransportConsole.exe from the <Installation Location>\Microsoft EDRA\CS\ReferenceArchitecture\Transports\Hosts\Console\bin\Debug directory and verify that only **MSMQDispatchingTransport** is running. **RemotingTransport** and **MSMQTransport** should not be running.
5. Run the Web application using the following URL:
http://localhost/GlobalBank/Default.aspx

To Uninstall

This section outlines the steps you should follow to uninstall the EDRI.

► To uninstall the EDRI

1. Run the UnInstall Reference Implementation.wsf script from the <Installation Location>\Microsoft EDRA\CS\ReferenceImplementation directory. This script removes the virtual directories that were created during the installation and removes all the databases that were installed. When the script runs, you will see the following output to the Setup console that launches.

```
Starting Enterprise Development Reference Implementation uninstall...
Removing the virtual directories...
Restarting IIS...
Running the database deletion script...
UnInstall Completed.
```

```
Creating the UnInstall completed flag file...
UnInstall Completed.
```

Check the Setup.log file for more details

Script finished. Press [ENTER] to close

2. Use **Add/Remove Programs** in **Control Panel** to uninstall the Enterprise Development Reference Implementation. Alternatively, you can uninstall by navigating to the location where you downloaded the .msi file, and then double-click **Enterprise Development Reference Implementation.msi**. In the **Welcome to the Enterprise Development Reference Implementation Setup Wizard** dialog box, select **Remove Enterprise Development Reference Implementation**, and then click **Finish**.

3. Check that the Global Bank databases installed by the EDRI were deleted. The databases are the following:

- GlobalBank_Core
- GlobalBank_CreditCard
- GlobalBank_CumulativeDeposit
- GlobalBank_InvestmentFund

If the SQL Server process is stopped, you will not be able to drop the database as you will not be able to connect to the SQL Server. SQL Server doesn't allow you to drop the database if there are any existing connections to the database. SQL Enterprise Manager might have a connection to the database if you are browsing it, or Query Analyzer might, or IIS if it has a cached connection, and so on. These processes must be stopped with the main SQL Server process left running for the database to get dropped successfully by the uninstall script.

4. Check that the following folders were removed from the C:\Documents and Settings\<Login Account>\VSWebCache\<Machine Name> folder:
 - AccountStatementService
 - AuthenticationService
 - BillPaymentService
 - BillSubscriptionService
 - ExternalInvestmentFundSystemService
 - FundsTransferService
 - GlobalBank
 - TransactionLogService
5. Delete the <Installation Location>\Microsoft EDRA\CS\ReferenceImplementation directory. This ensures that all files created during the build and test process are removed.

Troubleshooting

This section provides guidance on troubleshooting Web service projects and the EDRI set up.

Troubleshooting Web Service Projects

If you have problems with Web service projects when opening the GlobalBank solution in Visual Studio .NET, clear the VSWebCache.

► To clear the VSWebCache

- Delete the following folders from
C:\Documents and Settings\<login account>\VSWebCache\<Machine Name>:
 - AccountStatementService
 - AuthenticationService
 - BillPaymentService
 - BillSubscriptionService
 - ExternalInvestmentFundSystemService
 - FundsTransferService
 - GlobalBank
 - TransactionLogService

Troubleshooting Set Up

This section provides an alternate process for setting up the EDRI in case you experience any problems with the Build and Deploy Reference Implementation.wsf script. You should first follow the installation instructions in the “Installing the Enterprise Development Reference Implementation” section.

► To set up the EDRI

1. Run the Setup Reference Implementation.wsf script from the
<Installation Location>\Microsoft EDRA\CS\ReferenceImplementation directory. When the script runs, the Setup console prompts you to confirm whether the software prerequisites are installed. Click **Yes** if you have verified this to be true. Otherwise, click **No**.
2. The script prompts you to enter the SQL Server instance name. The default instance name is **(local)**. If the SQL Server database you are using is MSDE, you need to enter the instance name, instead of (local), in the text box in the following format: *server_name\instance_name*. In most cases, this is the instance name of the database, such as the machine name.

3. When the script runs, you will see the following output to the Setup console that launches.

```
Starting Enterprise Development Reference Implementation setup and deploy...
Starting Enterprise Development Reference Architecture setup...
Compiling Services Reference Architecture...
Setting up Business Event Handler...
Setting up Transaction Handler...
Setting up Performance Counters...
Setting up Event Log Source...
Setting up Logging Application Block...
Starting database setup...
Creating the setup completed flag file...
Setup Completed.
```

Check the Setup.log file for more details

Script finished. Press [ENTER] to close

4. On the taskbar, click **Start**, point to **Programs**, point to **Microsoft patterns & practices**, point to **Enterprise Development Reference Implementation**, and then click **GlobalBank C#**. Alternatively, you can navigate to the <Installation Location>\Microsoft EDRA\CS\ReferenceImplementation directory, and then double-click **GlobalBank.sln**.
5. To build the solution; in Visual Studio .NET, click **Build Solution** on the **Build** menu, or press CTRL+SHIFT+B.
6. Using the Visual Studio .NET command prompt, go to the <Installation Location>\Microsoft EDRA\CS\ReferenceImplementation directory and run the CreateDatabaseAndPopulateDataForReferenceImplementation.cmd script with **debug** or **release** as the parameter. This script sets up four databases and creates the appropriate accounts for accessing each database. For example, run the script with the following line.

```
CreateDatabaseAndPopulateDataForReferenceImplementation.cmd debug
```

When the script runs, you will see the following output to the Setup console that launches.

```
Copying the UserNamePasswordUtility files to the local folder...
    1 file(s) copied.
    1 file(s) copied.
Creating the GlobalBank_Core database...
Creating the GlobalBank_CreditCard database...
Creating the GlobalBank_CumulativeDeposit database...
Creating the GlobalBank_InvestmentFund database...
Populating database with data...
Populating database with base data...
```

(continued)

(continued)

```
Populating database with customer data...
Updating User Name and Password...
Removing the UserNamePasswordUtility files from the folder...
```

Script execution is complete!

7. Run the Deploy Reference Implementation.wsf script from the <Installation Location>\Microsoft EDRA\CS\ReferenceImplementation directory. When the script runs, you will see the following output to the Setup console that launches.

```
Starting Enterprise Development Reference Implementation Deploy...
Starting config files update...
Starting Transport config files update...
Starting Web Application config file update...
Starting deployment of the binaries...
Creating the setup completed flag file...
Setup Completed.
```

Check the Setup.log file for more details

Script finished. Press [ENTER] to close

You should now be able to run the Web application using the following URL:

<http://localhost/GlobalBank/Default.aspx>

You can log on to Global Bank using the Online ID and password listed in the “Setting Up and Running the EDRI” section earlier in this chapter.

More Information

For information about how to configure Microsoft SQL Server to store ASP.NET session state, see:

HOW TO: Configure SQL Server to Store ASP.NET Session State

<http://support.microsoft.com/default.aspx?kbid=317604>

The remaining references in this section are to *Building Secure Microsoft ASP.NET Applications: Authentication, Authorization, and Secure Communication*, Redmond: Microsoft Press, 2003, ISBN: 0735618909. The material can be read online at the following MSDN locations.

For information about securing the data transported between the client browser and the Presentation tier (Web server), see:

How To: Set Up SSL on a Web Server

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/SecNetHT16.asp>

For information about securing the data transported between the Presentation tier and the Application tier, see:

- “The Trusted Subsystem Model” in Chapter 3: Authentication and Authorization
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/SecNetch03.asp>
- How To: Use IPSec to Provide Secure Communication Between Two Servers
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/SecNetHT18.asp>
- How To: Use SSL to Secure Communication with SQL Server 2000
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/SecNetHT19.asp>

For information about securing the data transported between the Application tier and the Data tier, see:

- Accessing SQL Server Using Windows Integrated Security
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/html/vbtskaccessingsqlserverusingwindowsintegratedsecurity.asp>
- How To: Use IPSec to Provide Secure Communication Between Two Servers
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/SecNetHT18.asp>
- How To: Use SSL to Secure Communication with SQL Server 2000
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/SecNetHT19.asp>

Appendix A

Inside the Enterprise Development Application Framework

Introduction

The Enterprise Development Reference Implementation uses the Enterprise Development Application Framework to standardize the development of services. This appendix includes extracts from the “Enterprise Development Application Framework Architecture” section to help you become familiar with the EDAF. See the [Enterprise Development Reference Architecture](#) for more information.

Architectural Goals and Prerequisites

The Enterprise Development Application Framework was developed to meet a number of key architectural requirements and goals. These goals have had a major impact on the framework design.

Goals

Organizations transitioning from tightly coupled application architectures to loosely coupled, service-based architectures face numerous challenges. The more significant challenges include:

- Separating the service interface from the internal service implementation. This separation allows an enterprise to develop a custom deployment scenario that is optimized for scalability, reliability, security, performance, and availability.
- Separating business logic from cross-cutting concerns such as logging, monitoring, or raising business events.
- Separating business logic from the underlying transport so that multiple transports can be used to access a single service implementation.

The Enterprise Development Application Framework simplifies the development of distributed applications. It does this by providing standardized, reusable solutions to common challenges that software professionals face when developing and exposing business services.

For the purposes of the EDAF, these challenges were refined into the following architectural requirements:

- Provide support for sending service requests and receiving service responses over multiple transports, including Web services, message queuing, and remoting.
- Provide a declarative mechanism for specifying and applying cross-cutting concerns to a service across multiple transports.
- Separate service interface and service implementation elements so that they can run on different hosts.
- Allow business logic to be dispatched on Internet Information Service (IIS), COM+, or Microsoft Windows[®] service-based hosts, independent of the interface that transport requests came on.
- Separate business logic from transaction logic; treat transactions as cross-cutting concerns.
- Provide a simple means of integration with Microsoft BizTalk[®] server orchestration by using Microsoft Message Queuing.

EDAF at a Glance

The interaction between a client application and the service (or business action) it invokes can be simplified as a request and a response. Within the EDAF, the *business action* refers to the requested service and excludes everything but the actual business logic. In other words, it is the component that performs the business logic requested by the client application. Figure 1 illustrates this simple concept.

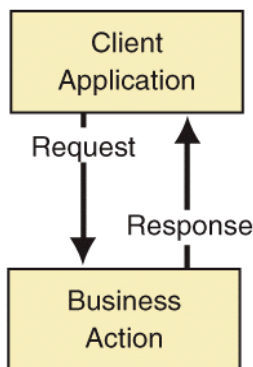


Figure 1

Simple service request from client application to business action

There is more involved in this process than simply sending a request to a business action and waiting for the response. For example:

- How will the message be delivered?
- Is the request sent across process boundaries or server boundaries?
- How will errors be handled?
- Will transactions be supported?

Considerations such as these are just a few of many that can complicate this interaction. To solve these complexities and make the interaction simpler for both the client application and the business action, the EDAF addresses these cross-cutting concerns. It does this by intercepting the request and processing it before sending it on to the business action. Figure 2 demonstrates how this works.

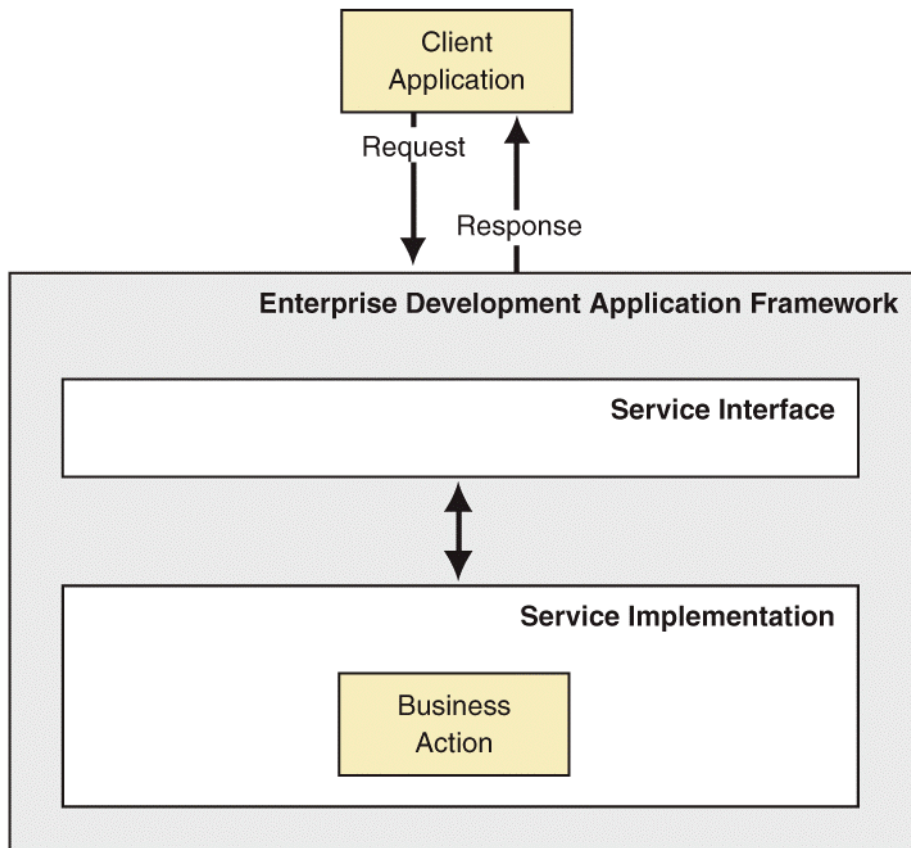


Figure 2

Client application invokes the business action using the Enterprise Development Application Framework

Conceptually, the EDAF sits between the client application and the business action. Within the EDAF, you find two layers: the Service Interface layer, and the Service Implementation layer. The Service Interface layer serves as a boundary, rejecting unauthorized or invalid requests. The Service Implementation layer is responsible for invoking the business action.

Figure 3 provides a high-level conceptual view of the key elements of the EDAF.

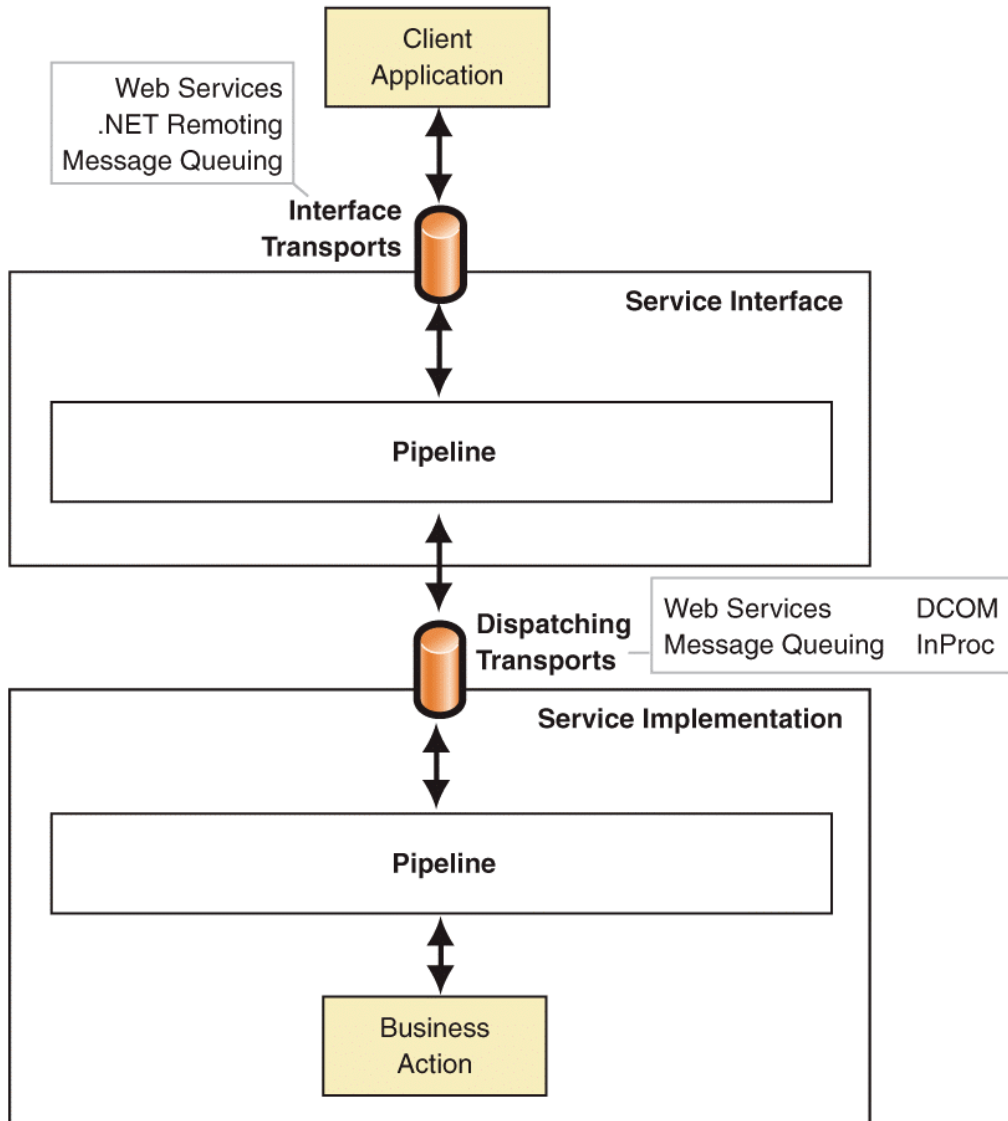


Figure 3

A high-level look inside the Enterprise Development Application Framework

The architectural goals of the framework include the ability for client applications to invoke components over multiple transports. Figure 3 illustrates how this works.

1. A client application sends a request for a service using one of the available transports, such as a Web service, a message queue, an InProc, or a Microsoft .NET Remoting component.
2. The transport receives the request and passes it to an instance of the Service Interface pipeline.
3. The pipeline applies handlers to the incoming request, passes the request to the pipeline target, which takes some action, and then applies the remaining handlers to the response received from the target. Each handler implements its logic and is executed in turn.

In the EDAF, the first pipeline (the Service Interface pipeline) is transport-specific and focused on service boundary handlers. These handlers exist at the periphery of the EDAF and perform initial checks and validations. For example, they request authentication and perform message request validation. The target of the Service Interface pipeline is another transport. This transport is responsible for invoking the second pipeline (the Service Implementation pipeline).

The Service Implementation pipeline is service-specific and focuses on business-related handlers. These handlers raise business events, log requests, initiate transactions, and so on. The target of the Service Implementation pipeline is responsible for invoking the business action.

A business action is the actual implementation of the requested service. It is either an internal business component or some other component that can call a business component(s). In either case, the business action is the ultimate target of the original service request from the client application. In most cases, when the business action is executed, it produces a response. The response is returned to the client application as follows:

1. When the business action is completed, it issues a response.
2. The Service Implementation pipeline receives the response, applies its remaining handlers, and returns the response to the Service Interface pipeline.
3. The Service Interface pipeline executes its remaining handlers, and sends the response to the transport.
4. The transport returns the response to the client application.

Dominant Patterns

Three patterns have dominated the architecture of the framework:

- The Service Interface pattern, as described in *Enterprise Solution Patterns Using Microsoft .NET*, Redmond: Microsoft Press®, 2003
- A variant of the Interceptor pattern called the Delegator pattern, as described in *Pattern-Oriented Software Architecture, Volume 2, Patterns for Concurrent and Networked Objects*, by Douglas Schmidt et al., Chichester, England: John Wiley & Sons, 2000
- The Chain of Responsibility pattern, as described in *Design Patterns: Elements of Reusable Object-Oriented Software*, by Erich Gamma et al., Reading, Massachusetts: Addison Wesley Longman, Inc., 1995

The Service Interface pattern decouples service interface mechanisms from the service implementation, so that they can vary independently. In the framework, the service implementation is represented by business action objects. Business action objects are invoked by the service implementation elements of the framework and serve as entry points to service business logic.

Service interface elements perform boundary functions for the service. They receive service requests from clients, dispatch those requests to service implementation elements, and return results to clients. The framework provides multiple mechanisms to allow independent evolution of the service interface and the service implementation. It also allows for multiple ways of hosting and deploying service interface and service implementation elements, either together or separately.

The Delegator pattern provides a request interception mechanism that allows cross-cutting logic to be inserted into the request processing flow. The interceptor takes the form of two pipelines as shown in Figure 4. The primary reason for using two distinct pipelines is to enable deployment scenarios where they are on separate computers for reasons such as security or reliability.

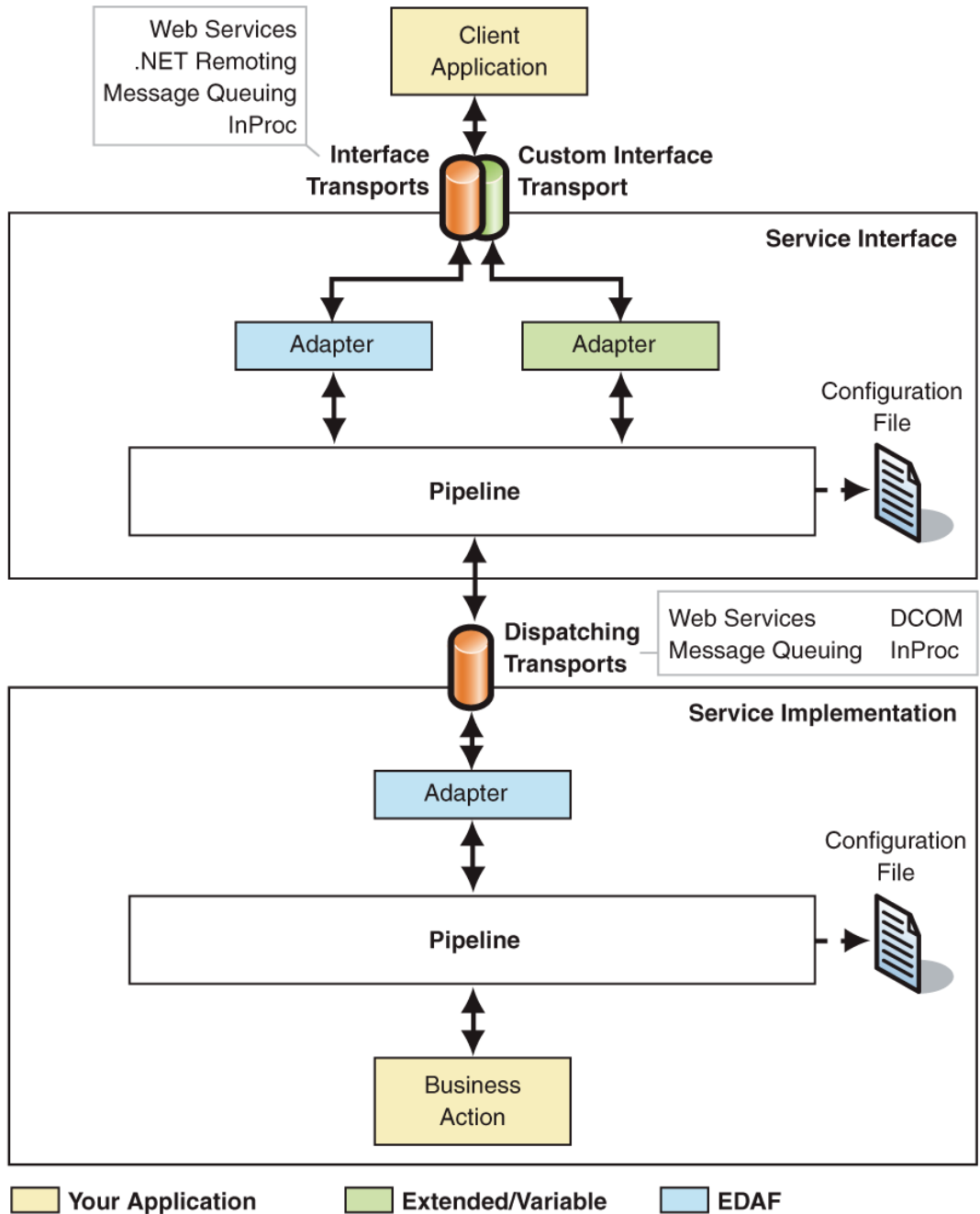


Figure 4
EDAF implementation of the Delegator pattern

The cross-cutting logic is included in the pipelines as handler objects. The same pipeline class (type) is used for all service interface transports. Each time a service request message arrives, an interface transport adapter places the message into a common envelope object (the **Context** object), and passes it to a pipeline instance. The pipeline instance uses configuration data supplied by the adapter to identify:

- The handlers to be invoked on the inbound service request message.
- The pipeline target to be invoked when the request reaches the end of the pipeline.
- The handlers to be invoked when the service response message is returned.

As described above, the framework has separate pipelines for the Service Interface and Service Implementation layers. The Service Interface pipeline is transport-specific (it is used for Web services, Message Queuing, InProc, or .NET Remoting), and focuses on service boundary handlers. Service boundary handlers usually perform initial checks and validations, such as authentication, request monitoring, and message request validation. The target of this pipeline is the *dispatching* target, which is responsible for invoking the second pipeline in the service implementation. This approach is known as a Bridge pattern. (For more information about dispatching targets, see “Service Request Flow” in the next section.)

The pipeline in the Service Implementation layer (the *implementation pipeline*) is service-specific and focuses on service-specific behavior. Handlers in this pipeline are usually related to and focused on business actions. These handlers raise business events, log service requests, or manage transactions. The target of an implementation pipeline is responsible for invoking a business action.

A business action is the entry point into the implementation of the requested service. It is either a business component, or it calls your business component(s). In either case, this is the ultimate target of the original service request. A business action is executed and, in most cases, produces a response.

The Chain of Responsibility pattern is used to guide the design pipelines. In the framework, pipelines assemble handlers in a chain of responsibility, as shown in Figure 5.

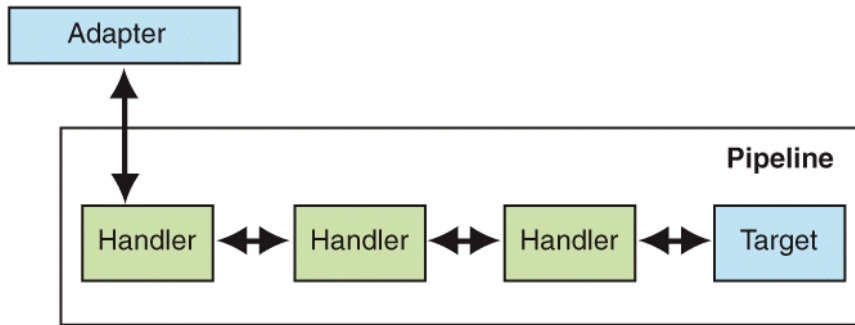


Figure 5

The Chain of Responsibility pattern

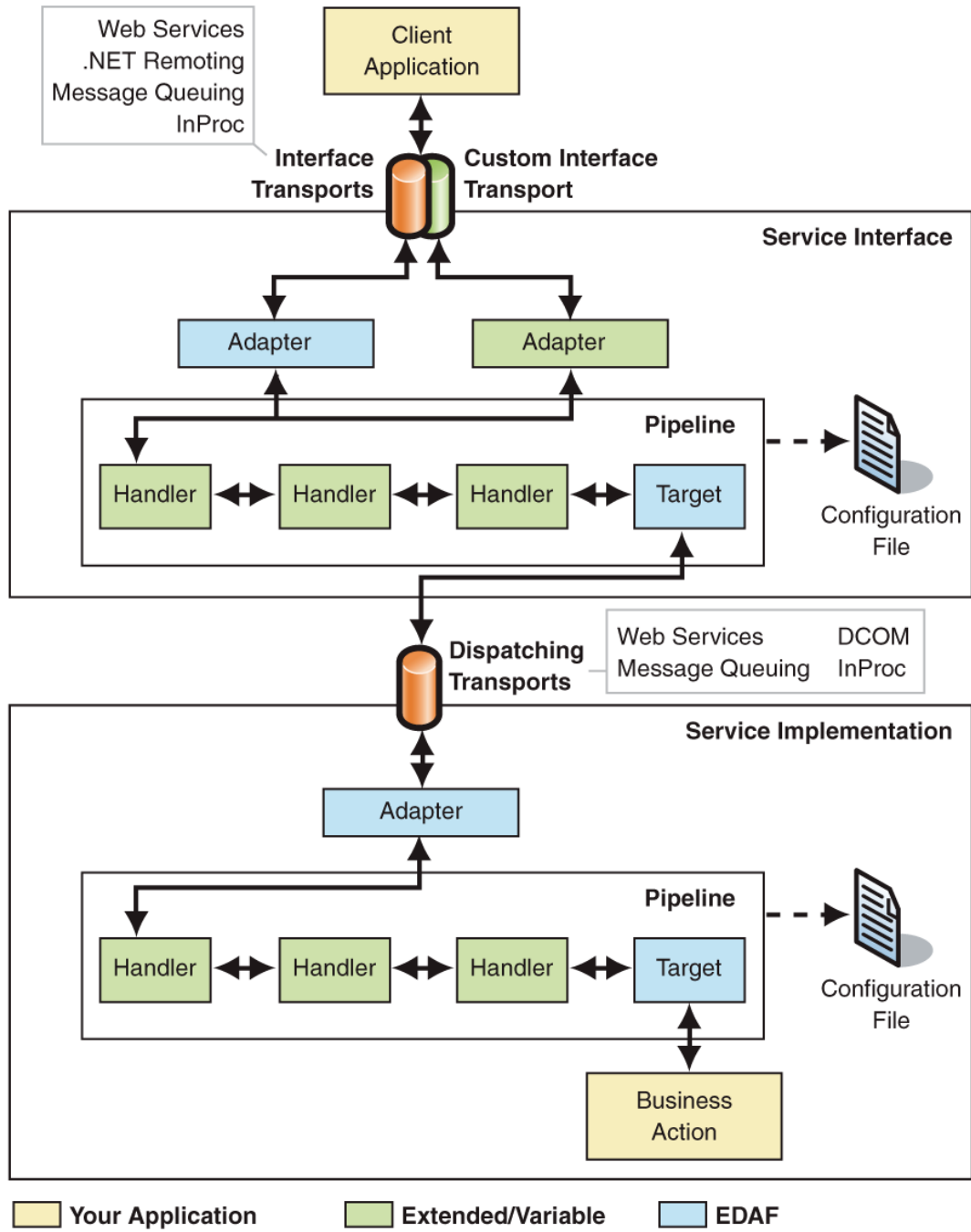
For a description of the details of pipeline design, see “Pipelines” in Chapter 3, “Logical View” in the EDRA documentation.

The pipeline constructor uses configuration data to stack handler objects in such a way that the pipeline calls the first handler, and each subsequent handler calls the next handler. The last element in the chain is the pipeline target, which reverses the process, returning control to the last handler, which then returns control to the handler that called it, and so on. This design allows handlers to act on requests and on the responses that are returned. Not all handlers take advantage of this feature. Handlers that process both requests and responses are referred to as *around* and *stacked around* handlers. Handlers that handle only requests or responses are referred to as *atomic* handlers.

Service Request Flow

To complete the conceptual view of the framework, you should understand the flow of service request from client to business action and the flow of the response back to the client. This section explains the request flow, reiterating key processes and concepts. In addition, Figure 6 provides a high-level representation of the service request flow from start to finish.

Note: The framework also supports a one-way service request flow (the fire-and-forget-it request model), in which no result is returned to the client.

**Figure 6***The EDAF process flow*

In this process, the client application creates a service request message and submits it by sending the message over the appropriate transport to the server that hosts the service interface.

Each transport is associated with a host. For example, the Web service transport is hosted in IIS, while most Message Queuing and remoting transports are hosted in a Windows service. A transport has an adapter that receives the message and initiates message processing. It creates a **Context** object, inserts the message into the object, creates an interface pipeline instance, and then passes the context to the pipeline to be processed.

The pipeline consists of a chain of handlers followed by a target object. As the control moves from handler to handler, each handler has access to the **Context** object and uses it to carry out its operation.

Pipelines are configurable in that different handlers can be chained within them. They are also configurable, in terms of which target they use. Interface pipeline targets (dispatching targets) implement different, transport-specific strategies when calling the implementation pipeline in the service implementation.

Dispatching targets are transport-specific in that they call implementation pipelines over specific transports. The framework provides supports for the following dispatching transports:

- Web service
- Inproc
- Message Queuing
- DCOM

Each dispatching transport has an adapter — similar to an interface transport adapter — that creates an instance of an implementation pipeline and passes the **Context** object to it.

The target of an implementation pipeline is a business action target. The target is responsible for calling the business action identified in the configuration data. The business action executes, places the results in a response message, adds the message to the **Context** object, and returns it to the business action target.

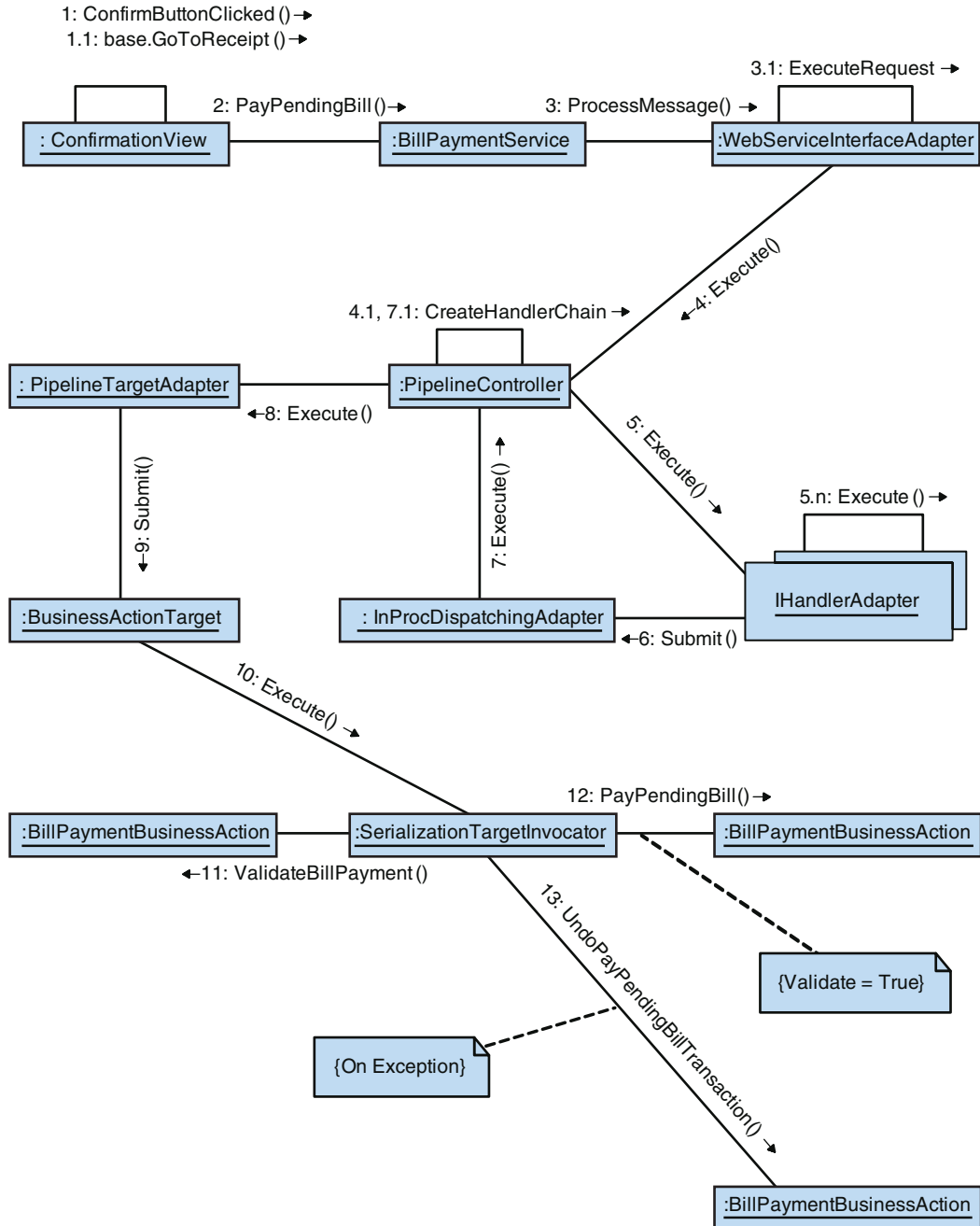
The target returns the context to the last handler in the pipeline chain that called it. This handler now has the **Context** object — with both the request message and the response message — and can perform an action on it before returning it to the adjacent handler in the stack. This process repeats until the first handler in the chain returns the **Context** object to the dispatching transport adapter.

The dispatching transport adapter returns the **Context** object by using its transport to the dispatching target of the interface pipeline, which returns it to the last handler, and so on. Finally, the interface transport adapter that initiated the request receives the **Context** object with request and response messages in it. Before returning the response to the client application, the adapter determines whether any exceptions have occurred. If an exception has occurred, the adapter establishes whether it was a business exception or infrastructure (framework) exception. Business exceptions are returned to the client. Infrastructure exceptions are caught, logged, and a safe error message is sent to the client with information about the exception that the system administrator can use to locate the exception details in a log.

Appendix B

Exploring the EDAF Using the Bill Payment Use Case

The collaboration diagram in Figure 1 will help you understand how the EDAF processes requests. Figure 1 shows the high-level interactions between client and framework components for the Bill Payment use case. The Pay Pending Bills use case is described because it covers many of the features exposed by the EDRI. Most of the other use cases are not as complex.

**Figure 1***Pay Pending Bill collaboration*

This collaboration diagram begins on the Confirmation page of the user interface and includes the following steps:

1. The user clicks the **Confirm** button.
 - 1.1 The **GoToReceipt** method on the base class is called. The base class for Confirmation view is the **PageController** class, which initializes and invokes the Web service.
2. The **PayPendingBill** method is executed on the Web service.
3. The **WebServiceInterfaceAdapter** is a Web service extension provided by the EDAF. The Web service extension intercepts and processes the requests coming to the service.
 - 3.1 An internal method named **ExecuteRequest** is used to process the request. This method also packages the message into a **Context** object and initializes the context with requested information.
4. The **WebServiceInterfaceAdapter** calls the **Execute** method on the **PipelineController**.
 - 4.1 The **PipelineController** uses information passed in the **Context** object to create a collection of handler adapters. The handler adapters are used to execute handlers as well as pipeline targets.
5. The **PipelineController** calls the **Execute** method on the first handler in the collection using the associated handler adapter.
 - 5.1 The **Execute** method is called on each handler in the collection for the pipeline associated with the invoked target. In this case, the target is the **InProcDispatchingAdapter**.
6. The **Submit** method is called on the **InProcDispatchingAdapter**. The adapter creates a new **Context** object using information from the original.
7. The **InProcDispatchingAdapter** calls the **Execute** method on the **PipelineController**. This **PipelineController** is associated with the Implementation pipeline.
 - 7.1 The **PipelineController** uses information passed in the **Context** object to create a collection of handler adapters. In this case, the only adapter is the **PipelineTargetAdapter**.
8. The **PipelineController** calls the **Execute** method on the **PipelineTargetAdapter**.
9. The **PipelineTargetAdapter** calls the **Submit** method on the **BusinessActionTarget**.
10. The **BusinessActionTarget** instantiates a target invocator based on the invocation type and calls the **Execute** method on that invocator instance. The target invocator used in this case is the **SerializationTargetInvocator** because the type of invocation is serialization. The **SerializationTargetInvocator** is responsible for instantiating the business action.

11. The appropriate validation method on the business action instance is called. In this case, the **ValidateBillPayment** method is called. This is called when the **validate** attribute on the business action is set to **true**.
12. If the validation is successful, the **PayPendingBill** method is invoked on the business action instance.
13. If there is an exception, the target invocator executes the method specified by the **compensate** attribute. In this case, it is the **UndoPayPendingBillTransaction** compensation method.

Now we can take a look at the handlers that the bill payment business action uses starting with the handlers in the Service Interface pipeline. The ordering of the handlers in the Service Interface pipeline is as follows:

1. **ExecutionTimeout** handler. First, we want to ensure the stability of the system by monitoring for hung threads from the beginning of the pipeline.
2. **SyntacticValidation** handler. Next, we will validate the request before doing any processing.
3. **Identity** handler. Authentication is another form of validation, so it makes sense to do it early. Also, it must happen before the **DuplicateMessage** handler.
4. **DuplicateMessage** handler. We use the **DuplicateMessage** handler with the mode set to "cache" for idempotency.

The ordering of the handlers in the Service Implementation pipeline is:

1. **LoggingHandler**. Because this is an application specific responsibility, we will place this handler in the Service Implementation pipeline.
2. **AppInstrumentation** handler. Because this is an application specific responsibility, we will place this handler in the Service Implementation pipeline.

In this case, the ordering of the handlers in the Service Implementation pipeline is not important. For the configuration of each handler, see the "BillPayment Service" section in Chapter 2, "Architecture."

Before looking at the collaboration diagrams, it is important to understand that each handler entry in the configuration file implements a separate handler adapter. When created, each handler adapter contains a reference to the handler it manages along with the configuration stage that the handler came from, which will be referred to as the handler stage in the following diagrams.

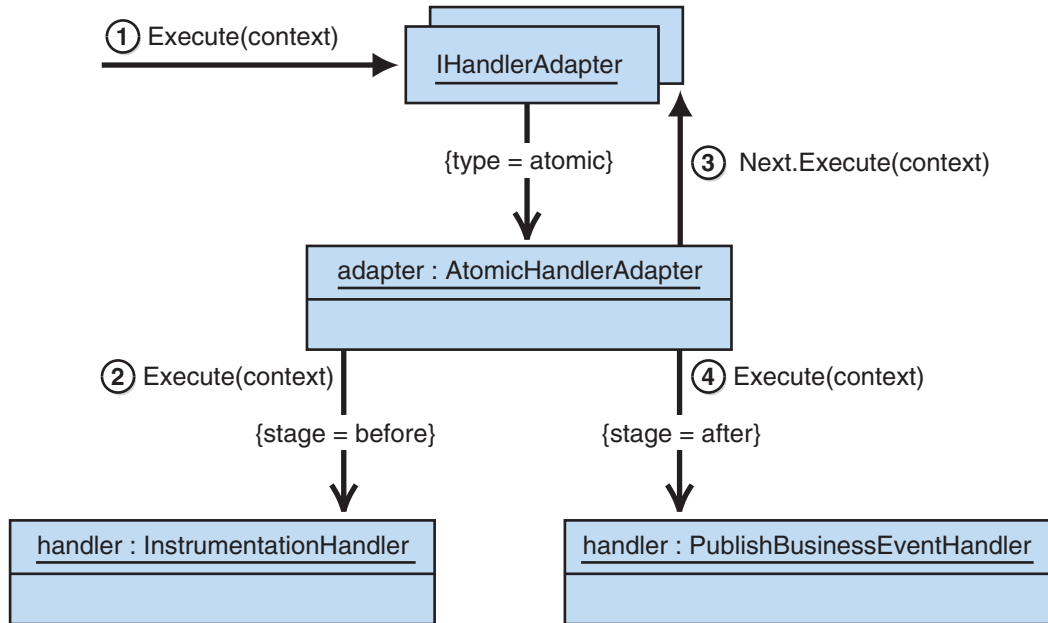
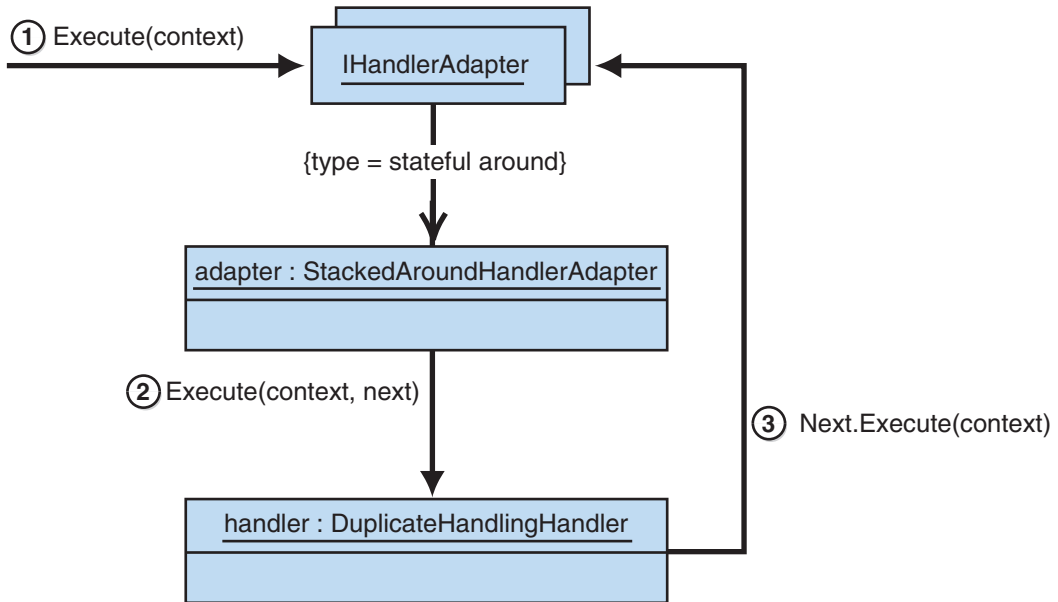


Figure 2

AtomicHandlerAdapter

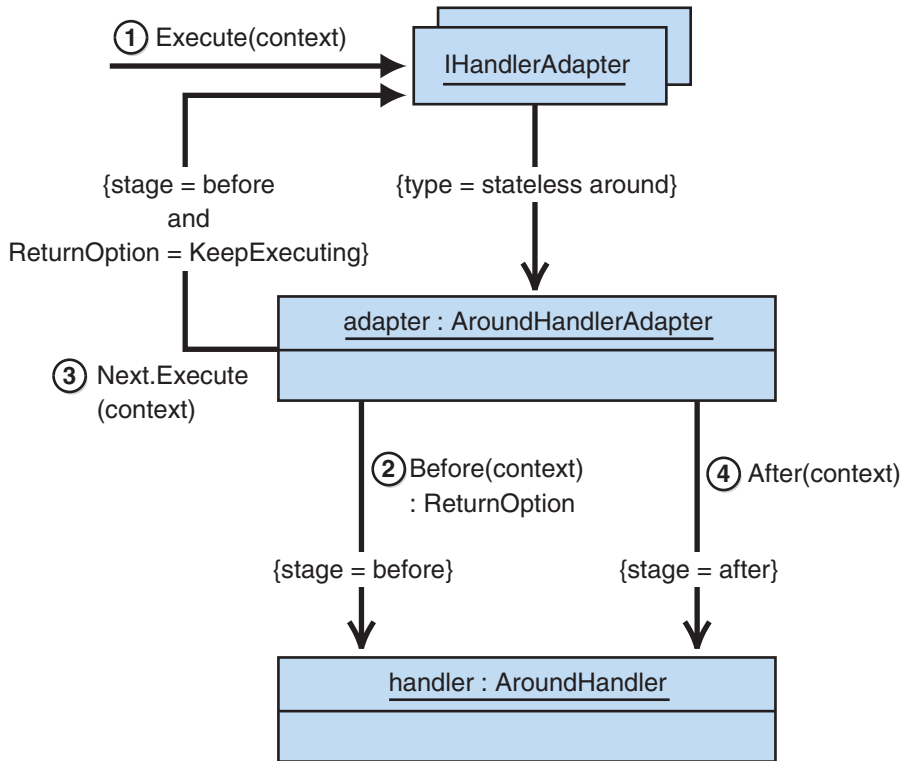
1. **Execute** is called on the first handler adapter in the collection.
2. If the adapter stage = before, then **Execute** is called on the handler instance.
3. The **Next** handler adapter is accessed and **Execute** is called.
4. If the adapter stage = after, then **Execute** is called on the handler instance.

The example above also shows two separate handlers, one that is called before the target and another that is called after the target. Each handler is called only one time, and each handler has access to the current context. During the before stage the context will only contain the request. During the after stage the context will contain both the request and response messages.

**Figure 3***StackedAroundHandlerAdapter*

1. **Execute** is called on the first handler adapter in the collection.
2. The adapter calls **Execute** on the handler, passing it the context and a delegate used to access the next handler adapter in the collection.
3. The next handler adapter is accessed, and **Execute** is called.

By calling the next adapter directly, the handler is able to process the request message prior to making that call, and then process the response once the next adapter returns control. In addition, the execution of all wrapped handlers and the target occur within this handler's execution call stack (and thread), which makes it possible to maintain state during those operations.

**Figure 4***AroundHandlerAdapter*

1. **Execute** is called on the first handler adapter in the collection.
2. If the handler stage = before, then the **Before** method of the handler is called.
3. If the handler stage = before and the **ReturnOption** = **KeepExecuting**, then the **Next** adapter is accessed and **Execute** is called.
4. If the handler stage = after, then the **After** method of the handler is called.

Similar to the atomic handler example, this example shows how two around handlers are treated as one. In reality there will be two handler adapters created, one for the before handler and another for the after handler. Each adapter is then initialized with the appropriate stage information.

When the adapter that contains the before handler is called, it will have access to the request only. When the adapter that contains the after handler is called, it will have access to both the request and response.

The **Before** method also returns a value that is used to indicate whether processing should continue or stop. When a value of **KeepExecuting** is returned, the adapter calls the next handler adapter. When a value of **StopExecuting** is returned, the next adapter is not called, and control is returned to the previous adapter or handler.

Contributors

The team that produced the Enterprise Development Reference Implementation came from a wide range of areas within Microsoft and from many of our partner organizations. The following people made a substantial contribution to the writing, developing, and testing of this content.

Program Management

Jason Hogg, Microsoft Corporation
Ron Jacobs, Microsoft Corporation
Sanjeev Garg, Satyam Computer Services

Architecture and Development

Naveen Yajaman, Microsoft Corporation
Jim Newkirk, Microsoft Corporation
Wojtek Kozaczynski, Microsoft Corporation
Jonathan Wanagel, Microsoft Corporation
Eugenio Pace, Microsoft Corporation
Alejandro Guillermo Jack, Southworks S.R.L.
Pedro Deviggiano, Southworks S.R.L.
Anil Jaswal, Infosys Technologies Ltd

Test

Edward Lafferty, Microsoft Corporation
Larry Brader, Microsoft Corporation
Sameer Tarey, Infosys Technologies Ltd
Mrinal Bhao, Infosys Technologies Ltd
Carlos Farre, Microsoft Corporation
V V Raviprasad, Infosys Technologies Ltd
Rohit Sharma, Infosys Technologies Ltd
Sireesha Tummala, Infosys Technologies Ltd
Mani Krishnaswami Subramanian, Infosys Technologies Ltd
Kaushik Barat, Infosys Technologies Ltd
Ganesh Gudar, Infosys Technologies Ltd
Rajul Vashistha, Infosys Technologies Ltd
Ravi Kant, Infosys Technologies Ltd
Prasanna S Raghavendra, Infosys Technologies Ltd
Ashima Aggarwal, Infosys Technologies Ltd
Ashok Reddy, Infosys Technologies Ltd

Mahesh Kumar, Infosys Technologies Ltd
Vani Gupta, Infosys Technologies Ltd
Sumit Kumar, Infosys Technologies Ltd
Pete Coupland, VMC Consulting Corporation
Scott Stender, @stake, Inc.
Andreas Junestam, @stake, Inc.
Kalyana Chakravarti Lakkamraju, Infosys Technologies Ltd
Ashok Bohra, Infosys Technologies Ltd
Harinarayan Paramasivan, Infosys Technologies Ltd

Documentation

RoAnn Corbisier, Microsoft Corporation
Sharon Smith Jansen, Microsoft Corporation
Lonnie Wall, RDA Corporation
Andrew Lader, RDA Corporation
Nelly Delgado, Wadeware LLC
Claudette Iebbiani, CI Design Studio
Brett R. Shriver, RDA Corporation
Tina Burden McGrayne, Entirenet
Erin Kilpatrick
Ping Kong

Review

Vijay Gajjala, Microsoft Corporation
Anna Liu, Microsoft Corporation
Andrew Mason, Microsoft Corporation
Stefan Schackow, Microsoft Corporation
Gandhi Swaminathan, Microsoft Corporation
Jim Williams, Microsoft Corporation
David Aiken, Arkitec
Brandon Bohling, Intel Corporation
Sergio A. Borronei
Vassilis Chazapis, Accenture
Michael Cleary
Benni DeMarco, FourThought Group Inc.
C. Bart Elia, Epicor Software Corporation
Adrie Geelhoed, LogicaCMG
Gerke Geurts, LogicaCMG
Christopher Haddix, Intel Corporation
Scott Hanselman, Corillian
Vaughn Hughes, Intel Corporation
Justin James, Intel Corporation
Gareth Jane, Ke Concepts

David Johnston, Intel Corporation
Boris Lublinsky
Marcus Mac Innes, StyleDesign.biz
Don Michie, Intel Corporation
Erymuzuan Mustapa, Inter Virtual Sdn. Bhd.
Henrik Natstrand, WM-data Consulting AB
Keith Organ, Arkitec
Mariano Omar Rodriguez
Roberto Schatz, Erbauer S.A.
JP Sklenka, LUCRUM Inc.
Hanspeter Stamm, Humm Computer Engineering AG
Randall Willis, Intel Corporation

patterns & practices

Mohammad Al-Sabt, Microsoft Corporation
Ward Cunningham, Microsoft Corporation
Scott Densmore, Microsoft Corporation
Matt Evans, Microsoft Corporation
Shaun Hayes, Microsoft Corporation
Sandy Khaund, Microsoft Corporation
Michael Kropp, Microsoft Corporation
William Loeffler, Microsoft Corporation
Rick Maguire, Microsoft Corporation
J.D. Meier, Microsoft Corporation
Per Vonge Nielsen, Microsoft Corporation
Ken Perilman, Microsoft Corporation
Juan Fernando Rivera, Microsoft Corporation
David Trowbridge, Microsoft Corporation
Srinath Vasireddy, Microsoft Corporation

patterns & practices

proven practices for predictable results

About Microsoft *patterns & practices*

Microsoft *patterns & practices* guides contain specific recommendations illustrating how to design, build, deploy, and operate architecturally sound solutions to challenging business and technical scenarios. They offer deep technical guidance based on real-world experience that goes far beyond white papers to help enterprise IT professionals, information workers, and developers quickly deliver sound solutions.

IT Professionals, information workers, and developers can choose from four types of *patterns & practices*:

- **Patterns**—Patterns are a consistent way of documenting solutions to commonly occurring problems. Patterns are available that address specific architecture, design, and implementation problems. Each pattern also has an associated GotDotNet Community.
- **Reference Architectures**—Reference Architectures are IT system-level architectures that address the business requirements, LifeCycle requirements, and technical constraints for commonly occurring scenarios. Reference Architectures focus on planning the architecture of IT systems.
- **Reference Building Blocks and IT Services**—References Building Blocks and IT Services are re-usable sub-system designs that address common technical challenges across a wide range of scenarios. Many include tested reference implementations to accelerate development. Reference Building Blocks and IT Services focus on the design and implementation of sub-systems.
- **Lifecycle Practices**—Lifecycle Practices provide guidance for tasks outside the scope of architecture and design such as deployment and operations in a production environment.

Patterns & practices guides are reviewed and approved by Microsoft engineering teams, consultants, Product Support Services, and by partners and customers. *Patterns & practices* guides are:

- **Proven**—They are based on field experience.
- **Authoritative**—They offer the best advice available.
- **Accurate**—They are technically validated and tested.
- **Actionable**—They provide the steps to success.
- **Relevant**—They address real-world problems based on customer scenarios.

Patterns & practices guides are designed to help IT professionals, information workers, and developers:

Reduce project cost

- Exploit the Microsoft engineering efforts to save time and money on your projects.
- Follow the Microsoft recommendations to lower your project risk and achieve predictable outcomes.

Increase confidence in solutions

- Build your solutions on proven Microsoft recommendations so you can have total confidence in your results.
- Rely on thoroughly tested and supported guidance, but production quality recommendations and code, not just samples.

Deliver strategic IT advantage






- Solve your problems today and take advantage of future Microsoft technologies with practical advice.

patterns & practices: Current Titles

October 2003

Title	Link to Online Version	Book
Patterns		
Enterprise Solution Patterns using Microsoft .NET	http://msdn.microsoft.com/practices/type/Patterns/Enterprise/default.asp	
Microsoft Data Patterns	http://msdn.microsoft.com/practices/type/Patterns/Data/default.asp	
Reference Architectures		
Application Architecture for .NET: Designing Applications and Services	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/distapp.asp	
Enterprise Notification Reference Architecture for Exchange 2000 Server	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnentdevgen/html/enraelp.asp	
Improving Web Application Security: Threats and Countermeasures	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/ThreatCounter.asp	
Microsoft Accelerator for Six Sigma	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/mso/sixsigma/default.asp	
Microsoft Active Directory Branch Office Guide: Volume 1: Planning	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/ad/windows2000/deploy/adguide/default.asp	
Microsoft Active Directory Branch Office Series Volume 2: Deployment and Operations	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/ad/windows2000/deploy/adguide/default.asp	
Microsoft Content Integration Pack for Content Management Server 2001 and SharePoint Portal Server 2001	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncip/html/cip.asp	
Microsoft Exchange 2000 Server Hosting Series Volume 1: Planning	Online Version not available	
Microsoft Exchange 2000 Server Hosting Series Volume 2: Deployment	Online Version not available	

To learn more about *patterns & practices* visit: <http://msdn.microsoft.com/practices>
 To purchase *patterns & practices* guides visit: <http://shop.microsoft.com/practices>

Title	Link to Online Version	Book
Microsoft Exchange 2000 Server Upgrade Series Volume 1: Planning	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/guide/default.asp	
Microsoft Exchange 2000 Server Upgrade Series Volume 2: Deployment	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/guide/default.asp	
Microsoft Solution for Intranets	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/mso/msi/Default.asp	
Microsoft Solution for Securing Wireless LANs	http://www.microsoft.com/downloads/details.aspx?FamilyId=CDB639B3-010B-47E7-B234-A27CDA291DAD&displaylang=en	
Microsoft Systems Architecture—Enterprise Data Center	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/edc/Default.asp	
Microsoft Systems Architecture—Internet Data Center	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/idc/default.asp	
The Enterprise Project Management Solution	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/mso/epm/default.asp	
UNIX Application Migration Guide	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnucmg/html/ucmglp.asp	
Reference Building Blocks and IT Services		
.NET Data Access Architecture Guide	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/daag.asp	
Application Updater Application Block	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/updater.asp	
Asynchronous Invocation Application Block	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/paiblock.asp	
Authentication in ASP.NET: .NET Security Guidance	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/authaspdotnet.asp	
Building Interoperable Web Services: WS-I Basic Profile 1.0	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsvcenter/html/wsi-bp_msdn_landingpage.asp	
Building Secure ASP.NET Applications: Authentication, Authorization, and Secure Communication	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/secnetlpMSDN.asp	

To learn more about *patterns & practices* visit: <http://msdn.microsoft.com/practices>
To purchase *patterns & practices* guides visit: <http://shop.microsoft.com/practices>

Title	Link to Online Version	Book
Caching Application Block	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/Cachingblock.asp	
Caching Architecture Guide for .Net Framework Applications	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/CachingArch.asp?frame=true	
Configuration Management Application Block	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/cmab.asp	
Data Access Application Block for .NET	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/daab-rm.asp	
Designing Application-Managed Authorization	http://msdn.microsoft.com/library/?url=/library/en-us/dnbda/html/damaz.asp	
Designing Data Tier Components and Passing Data Through Tiers	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/BOAGag.asp	
Exception Management Application Block for .NET	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/emab-rm.asp	
Exception Management Architecture Guide	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/exceptdotnet.asp	
Microsoft .NET/COM Migration and Interoperability	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/cominterop.asp	
Microsoft Windows Server 2003 Security Guide	http://www.microsoft.com/downloads/details.aspx?FamilyId=8A2643C1-0685-4D89-B655-521EA6C7B4DB&displaylang=en	
Monitoring in .NET Distributed Application Design	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/monitordotnet.asp	
New Application Installation using Systems Management Server	http://www.microsoft.com/business/reducecosts/efficiency/manageability/application.mspix	
Patch Management using Microsoft Systems Management Server - Operations Guide	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/msm/swdist/pmsms/pmsmsog.asp	
Patch Management Using Microsoft Software Update Services - Operations Guide	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/msm/swdist/pmsus/pmsusog.asp	
Service Aggregation Application Block	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/serviceagg.asp	
Service Monitoring and Control using Microsoft Operations Manager	http://www.microsoft.com/business/reducecosts/efficiency/manageability/monitoring.mspix	

Title	Link to Online Version	Book
User Interface Process Application Block	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/uip.asp	
Web Service Façade for Legacy Applications	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/wsfacadelegacyapp.asp	
Lifecycle Practices		
Backup and Restore for Internet Data Center	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/ittasks/maintain/backuprest/Default.asp	
Deploying .NET Applications: Lifecycle Guide	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/DALGRoadmap.asp	
Microsoft Exchange 2000 Server Operations Guide	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/exchange/exchange2000/maintain/operate/opsguide/default.asp	
Microsoft SQL Server 2000 High Availability Series: Volume 1: Planning	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/sql/deploy/confeat/sqlha/SQLHALP.asp	
Microsoft SQL Server 2000 High Availability Series: Volume 2: Deployment	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/sql/deploy/confeat/sqlha/SQLHALP.asp	
Microsoft SQL Server 2000 Operations Guide	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/sql/maintain/operate/opsguide/default.asp	
Operating .NET-Based Applications	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/net/maintain/opnetapp/default.asp	
Production Debugging for .NET-Connected Applications	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/DBGrm.asp	
Security Operations for Microsoft Windows 2000 Server	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/prodtech/win2000/secwin2k/default.asp	
Security Operations Guide for Exchange 2000 Server	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/prodtech/mailexch/opsguide/default.asp	
Team Development with Visual Studio .NET and Visual SourceSafe	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/tdlg_rm.asp	



This title is available as a Book

To learn more about *patterns & practices* visit: <http://msdn.microsoft.com/practices>
To purchase *patterns & practices* guides visit: <http://shop.microsoft.com/practices>